Desenvolvimento móvel : Uma solução Android completa com Rad Studio

Clube Delphi

Ano 12 :: Edição 144 :: R$ 11,90

DevMedia

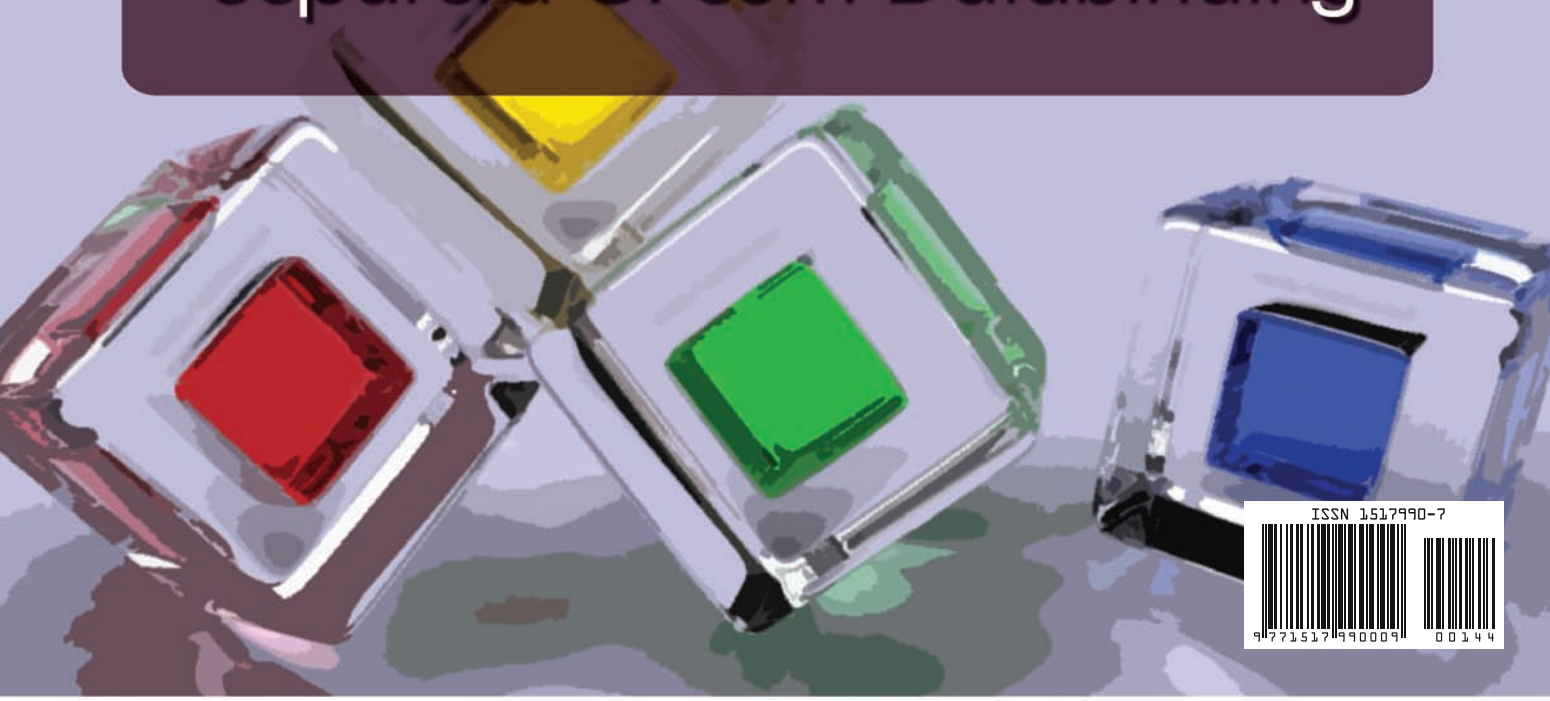
Streams
Faça um gerenciador de downloads multithread

InterBase XE
Fique por dentro do banco de dados da Embarcadero

CodeSite no Delphi XE 2
Depuração e criação de logs

MVVM

Separe a UI com Databinding



ISSN 1517990-7

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

Sumário

Boas Práticas

Boas Práticas

06 – MVVM no Delphi
[*Paulo Roberto Quicoli*]

Delphi

Novidade

16 – CodeSite no Delphi XE 2
[*Giuliano Scombatti Pinto*]

Banco de Dados, Minicurso

22 – InterBase XE – Parte 1
[*Fabrício Hissao Kawata*]

Boa Ideia

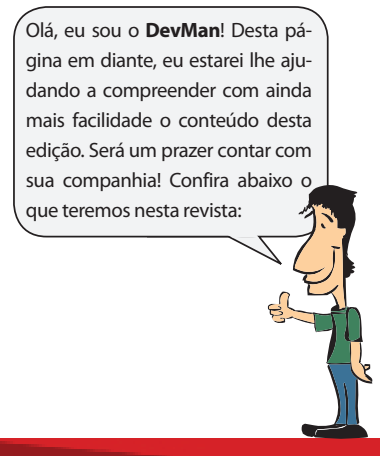
28 – Trabalhando com Streams e Threads
[*Giuliano Scombatti Pinto*]

Minicurso, Mobile

36 – Desenvolvimento Android – Parte 4
[*Giuliano Scombatti Pinto*]

Minicurso, Easy

44 – dbExpress de A-Z – Parte 3
[*Alexandre Borges*]



Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

www.devmedia.com.br/renovacao ou (21)3382-5038

DevMedia

Clube Delphi

Ano 12 - 144ª Edição 2012 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral

Paulo Quicoli
pauloquicoli@gmail.com

Editor Técnico

Daniel Sobrinho Laporte
daniel.laporte@gmail.com

Equipe Editorial

Guinther Pauli
guintherpauli@gmail.com

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br

Giuliano Scombatti Pinto
giuliano@sygnux.com.br

Jornalista Responsável

Kaline Dolabella - JP24185

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

EDITORIAL

Escrever um sistema que atenda aos requisitos de um cliente deve ser o objetivo principal de qualquer desenvolvedor, contudo, depois de pronto esse sistema pode ficar em funcionamento por anos. Ou seja, inicia-se a fase de manutenção do mesmo. Nessa fase sua empresa pode mostrar um diferencial à concorrência conseguindo responder às solicitações de mudanças de forma mais rápida e mais barata. Para conseguir isso um segundo objetivo deve ser incluído durante o desenvolvimento: a utilização de uma boa arquitetura. Quando falamos em arquitetura, a primeira visão que nos vem é a separação das camadas de um sistema, principalmente a visual e lógica de negócios. Nesta edição apresento um artigo sobre essa separação. Como aplicá-la em seus projetos utilizando um padrão de projeto pouco conhecido, o MVVM. Além disso, são explorados detalhes do framework que o implementa o DSharp e os benefícios obtidos ao se separar essas camadas.

O Giuliano traz três assuntos. Temos um artigo sobre o CodeSite da versão XE 2. Esse assunto é importante pois mostra como podemos armazenar um log de forma diferenciada, que poderá nos ajudar na fase de manutenção de um sistema. Ele também explora um estudo sobre o uso de Streams no Delphi, mostrando a criação de um gerenciador de download que faz uso de multithreads para quebrar o arquivo a ser baixado em três partes simultâneas. A aplicação desenvolvida expõe conceitos que podem ser reutilizados em outras situações, não perca. E nesta edição o minicurso sobre desenvolvimento Android se encerra, mostrando um pouco mais sobre jQuery. Espero que essa pequena série de artigos tenha despertado o interesse de muitos sobre o assunto, pois é o que movimentará o mercado nos próximos anos.

Toda a história entre InterBase e Firebird é conhecida pela maioria dos utilizadores do Firebird, contudo, uma vez que optaram pelo Firebird, nunca mais viram nada sobre o InterBase. O Fabrício mostra em seu artigo os recursos que banco de dados da Embarcadero possuir.

Por fim, o Alexandre traz a parte final sobre dbExpress, que os iniciantes agora se sintam confortáveis com a tecnologia e a utilizem eficientemente no seu trabalho diário.

Uma boa leitura!

Paulo Quicoli
pauloquicoli@gmail.com
Editor Geral
quicoli.wordpress.com
twitter.com/pauloquicoli

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DevMedia

Seu **conhecimento vale muito** na DevMedia

Publique artigos e vídeos gratuitamente no site e participe do programa **PayContent.**

Sua participação pode ser premiada: quanto mais você postar, mais chances de ganhar.

Para saber mais acesse:
www.devmedia.com.br/paycontent

DevMedia

MVVM no Delphi

Separe interface e regra de negócios

A separação da interface de usuário (UI) do restante das regras de negócio não é um assunto novo. Mesmo assim essa separação não é aplicada no dia a dia pela grande maioria dos desenvolvedores. Muitos pensam que para alcançar esse objetivo seria necessário alterar toda sua programação, abandonar o estilo RAD (Rapid Application Development) do Delphi e se voltar totalmente para a Orientação a Objetos.

É muito comum encontrar projetos onde os códigos para exibição de dados, sentenças SQL e validação de negócio estão centralizados nos formulários, esse é o famoso código espaguete ou macarrônico. Fazer um projeto dessa maneira é muito rápido e em tese, não é errado, pois o que importa é resolver o problema de um cliente. Mas além de pensar simplesmente nisso seria bom pensar também na vida desse projeto.

Existe uma fase no ciclo de vida de um projeto intitulada manutenção. Nela são implementadas melhorias, correções e até mesmo novos recursos. Essa fase pode fazer com que um projeto seja menos competitivo que outro ou até mesmo mais custoso. Neste caso, pode-se dizer ser menos competitivo porque dependendo de como foi escrito, alterar uma rotina pode levar dias, semanas. E consequentemente isso gera custo, o que torna cara a manutenção. O código macarrônico causa essa situação.

As camadas do projeto são amarradas entre si, gerando uma dependência desnecessária, impossibilitando a reutilização de rotinas, regras de negócio. Sem falar a aplicação de rotinas consagradas para atestar qualidade, como testes unitários.

Resumo DevMan

De que se trata o artigo:

Este artigo tem por finalidade destacar a importância de se organizar uma aplicação em camadas lógicas, como conseguir isso separando a interface visual de sua lógica ao fazer uso do padrão MVVM e como ele pode ser aplicado em Delphi através do Framework DSharp. São abordados conceitos como DataBinding, separação de camadas e as vantagens que essa separação traz.

Em que situação o tema é útil:

Este tema é utilizado para organizar o código fonte de uma aplicação, separando as funcionalidades existentes em camadas lógicas. Um exemplo do que deve ser separado é a camada visual e a camada de negócios, que uma vez independentes uma da outro, permitem a aplicação de outra boa prática, o uso de testes automatizados.

MVVM no Delphi:

Desde o Delphi 2010 vários recursos internos foram acrescidos à linguagem Delphi. Muitos deles localizados na parte de RTTI, ou introspecção de informações. Com isso a linguagem foi ficando poderosa o suficiente para facilitar o desenvolvimento de Frameworks que hoje auxiliam na aplicação das mais diversas boas práticas da orientação a objetos. A divisão de responsabilidades é uma dessas. Porque a camada de interface precisa estar tão ligada à camada de negócios? O ideal é que ela somente apresente os dados. Conseguir isso não era fácil, contudo temos agora o DSharp, um Framework que implementa o padrão de separação visual MVVM. O ponto chave desse padrão é a utilização do recurso chamado DataBinding, ou ligação de dados, para propiciar essa separação. É exatamente isso que o DSharp traz e que vamos explorar neste artigo, construindo exemplos e discutindo a teoria.

Arquitetura

Em um sistema com boa arquitetura encontramos algumas divisões lógicas que visam facilitar a manutenção e adição de novos recursos. Essas camadas podem ser vistas na **Figura 1**.

Para entender a funcionalidade de cada camada, vamos aplicar um exemplo comum, uma situação conhecida por todos, um formulário de cadastro. Esse formulário representa a camada de apresentação do sistema, isto porque é nela que os dados são apresentados ao usuário.

Em um sistema Cliente/Servidor, a sua camada de domínio, de base dados e até mesmo serviços podem ser representadas pelos Data Modules existentes e seus componentes de acesso ao banco de dados.

Por último, temos a camada global que pode ser representada pelo seu conjunto de funções e procedimentos públicos.

Continuando com o paralelo a um sistema comum, desenvolvido de forma macarrônica, nos formulários nós fazemos acesso direto aos Data Modules, aos ClientDataSets, geramos SQL, tudo sem preocupação alguma, ou seja, nós misturamos o que seriam nossas camadas sendo que o correto é separar as camadas.

Figura 1. Camadas lógicas de um software

Quando essas camadas se misturam é dito que existe um acoplamento entre elas. No momento que você insere código SQL em evento OnClick de um botão o acoplamento foi realizado. Seu formulário, que é sua camada de apresentação, está fazendo acesso direto aos recursos do que seria sua camada de dados. Isso impede a reutilização da lógica contida no formulário. Tudo isso dificulta a manutenção do seu sistema porque você terá SQL espalhado por todo código e não apenas centralizado em uma área.

Outra prática que é muito comum é de se criar os eventos e dentro deles escrever um código tão longo que para entender o que está sendo feito é quase como ler um livro. Consequentemente, a rotina ali dentro poderia ser quebrada em classes ou funções.

Quando uma rotina faz mais do que foi projetada a fazer é dito que existe uma baixa coesão nela. A baixa coesão é outro problema para a boa manutenção do seu Software. Então, como podemos separar as camadas sem evitando a baixa coesão e alto acoplamento?

Quem já está no mercado há algum tempo com certeza já ouviu sobre os padrões MVC (Model-View-Controller) ou MVP (Model-View-Presenter). Esses padrões de projeto, que são soluções atestadas por vários desenvolvedores, sugerem uma solução à separação da UI. Esses são padrões bem genéricos, que podem ser implementados por qualquer linguagem orientada a objetos, visto que não utilizam recursos específicos.

O MVC para separar a camada de apresentação cria três partes distintas, o Model, a View e o Controller. Podemos defini-los da seguinte forma: Model – É a representação da informação que a aplicação utiliza; View – É a representação gráfica do Model. Por exemplo, um formulário expondo as informações de um cliente. Uma View está sempre ligada a um model e sabe como exibi-lo ao usuário. Além disso, também pode ser composta por várias Views; Controller – É responsável em mapear as ações do usuário em ações do sistema, por exemplo, se um botão ou item de menu é clicado é ele que deve definir como a aplicação responderá. O MVP é por sua vez, uma extensão do MVC, tratando a View apenas como uma camada simples que implementa uma Interface contendo todas as propriedades e métodos necessários para o funcionamento da aplicação. Dessa maneira, se precisarmos incluir uma nova camada de apresentação, basta implementar a Interface em questão e a mesma funcionará da mesma maneira. Enquanto no MVC a View tem acesso ao Model e ao Controller, tornando assim a relação fortemente acoplada, no MVP a View "conhece" apenas o Presenter, possibilitando que o Presenter faça o "meio de campo" entre as demais camadas.

Existe outro padrão, chamado MVVM (Model-View-ViewModel), que explora recursos de tecnologia que nem sempre estão disponíveis nas linguagens e serão mencionados adiante. Esse padrão também é conhecido por Presentation Model.

MVVM/Presentation Model

O objetivo desse padrão é representar o estado e comportamento de uma View, independente dos controles utilizados. A essência do MVVM está na classe (View Model) que representa todos os dados e comportamentos de uma UI, mas sem conter os controles utilizados para exibir esses dados. A UI, representada por View, simplesmente projeta o estado do View Model. Para isso o View Model contém propriedades para todas as informações dinâmicas da View. Essas informações são compreendidas pelo conteúdo dos controles e valores de estado dos controles, utilizados, por exemplo, para habilitar ou não um controle visual. Isso não significa que seja necessário armazenar todo o estado de um controle visual, mas apenas aqueles que podem ser alterados pela interação com o usuário. Como o View Model deve conter os dados a serem exibidos, é ele que se conecta aos objetos de negócio, que são a parte chamada Model desse padrão e fica responsável por sincronizar com a View esses dados.

A principal diferença em relação ao MVC ou MVP é que as partes View e ViewModel não possuem dependência entre si. É possível construir a UI independentemente da regra de negócio e posteriormente ligar, ou usando o jargão específico: realizar o Binding dessas partes. Essa é palavra que faz esse padrão funcionar, Binding. Mas o que é um Binding?

Conceituando DataBinding

É um objeto que representa a ligação entre outros dois objetos. Ele define como dois objetos estarão ligados, sua relação. Em uma operação de Binding um objeto pode ser um Source ou Target (**Figura 2**). Source é o objeto "fonte de dados" enquanto que o Target é o alvo, o objeto que irá exibir o Source. Também é definido o que exibir e onde exibir, para isso são informadas a propriedades relativas de cada objeto. Por exemplo, podemos determinar que a propriedade Nome de um objeto TPessoa seja exibida na propriedade Text de um controle TEdit.

É possível ainda determinar o fluxo dos dados, ou seja, quem atualiza o que. Por exemplo, se apenas o Source é que envia dados para o Target, se apenas o Target envia ou ambos trocam informações. Com esse comportamento podemos determinar uma espécie de Binding somente leitura, onde um TEdit apenas apresenta os dados e não envia as alterações para o objeto source. Os fluxos possíveis podem ser vistos na **Figura 3**.

No modo OneWay somente o Source atualiza o Target. Já em TwoWay, os dois trocam informações e no OneWayToSource, o Target é que envia os dados para o Source. Nos modos em que o Source pode ser atualizado (TwoWay e OneWayToSource), podemos ainda determinar quando essa atualização é feita (**Figura 4**).

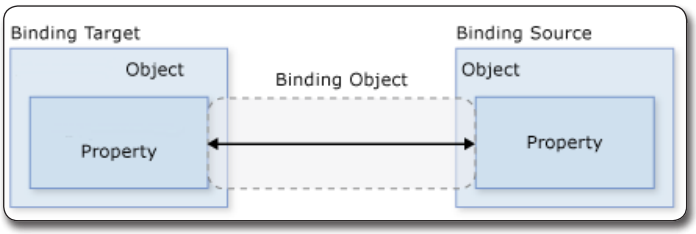


Figura 2. Relação de ligação

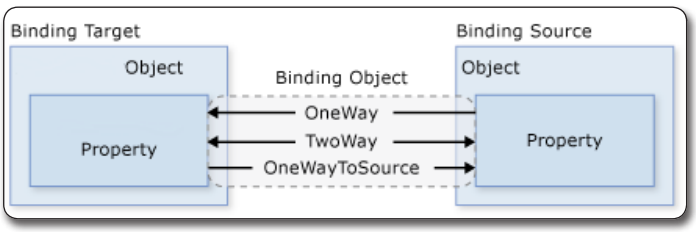


Figura 3. Fluxos de binding

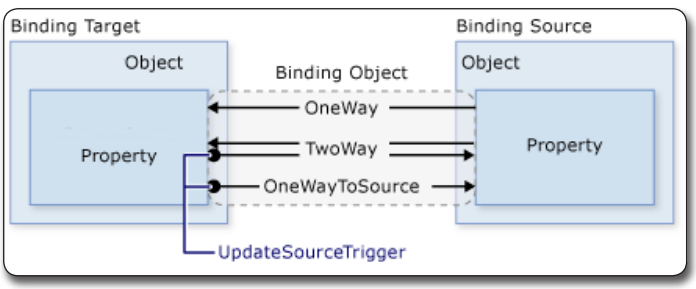


Figura 4. Determinando a atualização

Isso é feito através de uma propriedade UpdateSourceTrigger. Ela pode receber o valor LostFocus, onde a atualização ocorre quando o Target perde o foco e seu evento OnExit é disparado. O outro valor possível é o PropertyChanged, que atualiza o Source sempre que a propriedade do Target ligada ao Binding sofre alguma alteração. Por fim temos ainda o Explicit. Nesse caso o Target só envia as atualizações para o Source quando o desenvolvedor chamar o método UpdateSource do target.

Frameworks MVVM

Apesar do MVVM ser também conhecido como Presentation Model, na verdade ele é uma implementação específica que utiliza recursos do .NET/WPF como o mecanismo de Binding. Esse mecanismo permite ligar qualquer propriedade de um controle visual a uma propriedade ou método de objetos de negócio. Vários Frameworks foram desenvolvidos para explorar esse mecanismo da melhor forma e, com isso, agilizar a criação de um aplicativo que faça uso deste padrão.

Para Delphi existe o Framework DSharp. Ele foi desenvolvido observando o que existia para o .NET, extraindo conceitos que poderiam ser aplicados. Um desses é o chamado ModelView Binder, que representa uma classe que toma uma View (como um formulário) e um View Model (como um DataModule ou outro objeto de negócio) e pode criar ligações entre eles utilizando convenções nominativas. Por exemplo, no View Model temos uma propriedade chamada NomeCliente, na View temos um TEdit com o nome NomeCliente, automaticamente os dois são ligados entre si. A mesma ideia também é utilizada para preenchimento de controles como TComboBox, uma propriedade "Cidades" no View Model também poderia ser ligada a um combo Cidades da View. Objetos encadeados também podem ser ligados automaticamente obedecendo a um padrão. Digamos que no View Model temos Cliente.Endereco.Logradouro. Isso poderia ser ligado a um TEdit na View chamado Cliente_Endereco_Logradouro.

Essa forma de funcionamento é chamada de convenção sobre configuração ou simplesmente codificação por convenção. É um paradigma na modelagem de Software que tem o objetivo de fazer com que os desenvolvedores tomem menos decisões, gerando simplicidade, sem perder flexibilidade. Isso quer dizer que um desenvolvedor teria que se preocupar apenas com os aspectos não convencionais de sua aplicação. Se ele está codificando uma classe View Model chamada PessoasViewModel, a sua contrapartida View seria chamada PessoasView por padrão.

Quando temos um Framework que segue essas ideias, temos menos código de configuração a escrever. Outro conceito adotado pelo DSharp foi a adoção de um Framework de injeção de dependência. A injeção de dependência é um mecanismo onde é possível informar o que determinado objeto precisa para ser instanciado e esse Framework se encarrega disso. Por exemplo, temos um objeto TLog que recebe em seu construtor um parâmetro do tipo TTipoLog, que seria outra classe que contém a configuração para realização de um Log. Tudo isso implica que toda vez que fosse necessário criar um objeto TLog, também teríamos que criar um TTipoLog. Ao registrar essa dependência em Framework de injeção, simplesmente pedimos a esse Framework uma nova instância de TLog e ele se resolve para nos entregar. Esses são recursos bem avançados, contudo, neste artigo vamos focar no que é possível de fazer com Data Binding.

Tutorial

Instalando o DSharp

O Framework DSharp está disponível para Delphi 2010 e superiores. Neste artigo utilizaremos o Delphi XE 2. Após realizar o download de seu código fonte (veja seção **Links**) acessamos a pasta {pasta_instalação}\Packages\DelphiXE2 e carregamos o arquivo BuildPackages.groupproj. Esse arquivo representa o grupo de projetos que compõe o DSharp. Uma vez carregado, a janela Project Manager ficará como na **Figura 5**.

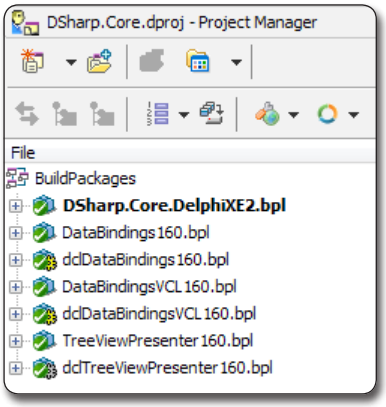


Figura 5. Projeto DSharp carregado

Com um clique com o botão direito do mouse sobre o projeto DSharp.Core.DelphiXE2.bpl, escolhemos a opção Build. Isso irá compilar o projeto. Fazemos o mesmo para DataBindings160.bpl, dclDataBindings160.bpl, DataBindingsVCL160.bpl. Para o projeto dclDataBindings160.bpl escolhemos a opção Install também. Isto porque esse contém um componente, o TBindGroup, que é instalado na guia DSharp. Os outros dois projetos (TreeView-Presenter160.bpl e dclTreeViewPresenter160.bpl) não precisam ser instalados pois referem-se a uma implementação específica para um componente chamado TVirtualStringTree, que não será utilizado neste artigo. Após a instalação incluímos no Library Path de cada pasta contida em {pasta_instalação}\Source, exceto as pastas DelphiWebScript, DevExpress e Testing.

Explorando o binding: entre controles

O Framework DSharp implementa todo o conceito de Binding apresentado. Para vermos como isso foi feito, criamos um novo projeto do tipo VCL Forms Application e lhe damos o nome de Demo1. No formulário criado adicionamos um CheckBox, com o Caption "Habilitar" e um TEdit. Vamos ver como realizar o Binding entre componentes visuais, ao marcar o CheckBox o TEdit ficará habilitado, ao desmarcar , o TEdit será desmarcado.

Esse Binding é controlado pelo objeto BindingGroup, então o mesmo também é adicionado ao formulário. Ao efetuarmos um clique duplo sobre o mesmo, surge o editor de Binding, é nele que vamos inserir os DataBindings. Ao clicar no botão Add New ou pressionar Insert, um novo objeto de Binding é criado e suas propriedades são expostas no Object Inspector (**Figura 6**).

Na opção BindingMode podemos configurar como será o fluxo de dados entre os objetos que serão conectados. Deixamos a opção padrão, bmTwoWay, marcada pois ela equivale ao conceito TwoWay explicado anteriormente, as propriedades Enabled e Managed vem ativas, elas respectivamente configuram a ativação do Binding e como será feita a atualização entre Target e Source. Se setarmos Enabled para false, o Binding fica inativo. Se setarmos Managed para false, nós teríamos que, manualmente, notificar ao Framework de Binding a mudança de valor no Source, através de comando BindingGroup1.UpdateTargets.

Cada objeto TBinding que é criado também pode disparar um evento quando o Source ou Target são atualizados. Isso é configurado nas propriedades NotifyOnSourceUpdate e Notify OnTargetUpdate.

Na propriedade Source podemos localizar o controle que será a fonte de dados. No caso escolhemos CheckBox1 e então podemos escolher qual propriedade dele estará no Binding. Em SourcePropertyName localizamos todas as propriedades públicas do controle selecionado em Source. Escolhemos então a propriedade Checked. Também é preciso configurar como será feita a atualização na propriedade do Source. Na propriedade SourceUpdateTrigger temos as opções utExplicit, utLostFocus e utPropertyChanged que correspondem ao que foi explicado pela **Figura 4**.

Em Target escolhemos qual controle que responderá a uma mudança do Source. Escolhemos então o Edit1. Também é preciso definir qual propriedade dele que estará ligada à propriedade Checked do CheckBox1. Em um primeiro momento podemos dizer que ambas as propriedades devem ter o mesmo tipo, assim, ligamos Edit1.Enabled à CheckBox1.Checked. Posteriormente veremos que também é possível realizar o Binding entre tipos diferentes.

Apenas com essas opções já temos uma ligação bem sucedida. O editor de Binding mostra de forma didática o Binding criado (**Figura 7**).

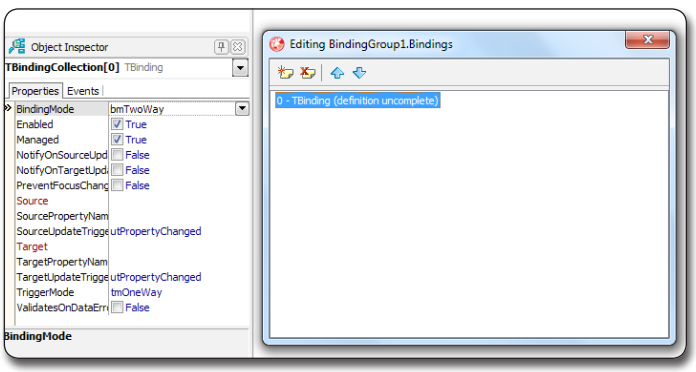


Figura 6. Criando um binding

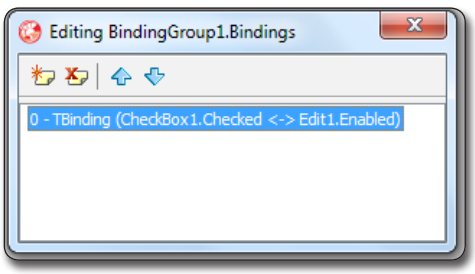


Figura 7. Binding criado

Facilmente identificamos quais objetos estão relacionados e como é esse relacionamento, expresso pelas setas direcionais.

Execute essa demonstração e veja que Edit1 só fica habilitado quando o CheckBox1 está marcado. Não escrevemos uma linha de código sequer para isso acontecer.

Nota

Para que as operações de Binding funcionem corretamente é preciso observar na seção uses a ordem das units. As units do DSharp devem ser referenciadas por último. Isso é necessário porque o Framework de Binding possui classes que "substituem" algumas classes padrão do Delphi. Por exemplo, existem implementações especiais em DSharp.Bindings.VCLControls que ativam o recurso de Binding em controles TEdit.

O Delphi quando carrega um arquivo DFM armazena referências para os controles declarados na seção uses, na ordem que as units estão declaradas, assim, se deixarmos as units do Framework no final, ele armazenará as versões especiais e não as padrões. Este é um "mal" necessário para que não tivéssemos que criar controles visuais apenas para realização de Binding.

Explorando o binding: conversores

Operações de Binding não são limitadas apenas a propriedades do mesmo tipo. É possível ligar coisas totalmente diferentes. No DSharp o conceito de ValueConverter foi implementado. Nele temos uma classe que é ligada ao objeto de Binding e este a usa para adaptar/converter um valor de um tipo para outro.

Vamos experimentar isso fazendo o seguinte: vamos adicionar um novo TEdit ao formulário e criar um novo Binding pelo editor de Binding. O Source será esse novo TEdit, a propriedade SourcePropertyName irá apontar para Text. O Target por sua vez será também o próprio novo TEdit, porém a propriedade TargetPropertyName irá apontar para Color. A **Figura 8** mostra essa configuração.

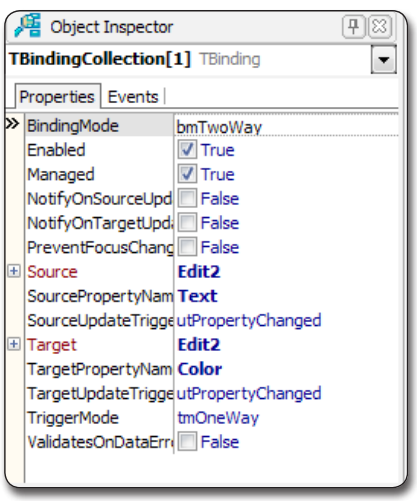


Figura 8. Propriedades do binding

A classe base para conversores chama-se TValueConventer e está localizada na unit DSharp.Core.DataConversion, assim a adicionamos na seção implementation. São definidos dois métodos, Convert e ConvertBack. O primeiro é utilizado pelo Framework no momento que o objeto Source recebe o valor e esse deve ser transferido para o Target. É nesse momento que a conversão é realizada. Já o ConvertBack é utilizado para fazer a operação inversa.

Dessa forma criamos na seção implementation a classe TIntToColorConverter que pode ser vista na **Listagem 1**.

Listagem 1. Classe TIntToColorConverter

```
01 TIntToColorConverter = class(TValueConverter)
02 public
03   function Convert(const Value: TValue): TValue; override;
04   function ConvertBack(const Value: TValue): TValue; override;
05 end;
06
07 function TIntToColorConverter.Convert(const Value: TValue): TValue;
09 var
10  valorInt: Integer;
11 begin
12  valorInt := StrToIntDef(Value.AsString,0);
13  if (valorInt > 0) then
14     Result := clGreen
15  else
16   if (valorInt = 0) then
17     Result := clWhite
18   else
19      Result := clRed ;
20 end;
21
22 function TIntToColorConverter.ConvertBack(const Value: TValue): TValue;
23 begin
24  raise Exception.Create('Não implementado');
25 end;
```

Como nossa classe conversora herda de TValueConverter, é necessário sobrecarregar a implementação base dos métodos Convert e ConvertBack usando o modificador override. Isso faz com que a implementação nossa seja executada e não a original. Quando a propriedade do Source receber um valor, o método Convert é chamado, recebendo em seu parâmetro Value o valor informado. Value é do tipo TValue que pertence à RTTI do Delphi e é utilizado para representar tipos, armazenando o valor e informações sobre o tipo do valor. Em sua definição já existem vários métodos de conversão, como AsString, AsInteger, AsInt64, AsBoolean e outros. Como pode ser visto na linha 12, usamos a função AsString para obter o valor contido em Value na forma de String. Então o convertemos para um inteiro. A regra de negócio implementada é bem simples: se o valor inteiro digitado no TEdit for maior que zero, retornamos a cor verde. Se for inferior retornamos vermelho e se for zero retornamos branco. Como o retorno do método também é do tipo TValue, simplesmente retornamos os tipos que representam as cores citadas.

Na linha 22 temos o método ConvertBack, como nossa intenção não é mudar a cor do TEdit, que é o Target, e sim devolver um número ao Source, simplesmente não temos uma implementação real, temos a criação de um exceção. Dessa forma se alguém tentar realizar algo que não queremos, o processo será interrompido.

Como já temos o conversor pronto, é preciso passá-lo ao objeto Binding criado anteriormente de forma visual.

Criamos então no formulário um método que ficará responsável por configurar as operações de Binding. Ele foi chamado de ConfiguraDataBinding e sua implementação é vista a seguir:

```
procedure TForm1.ConfiguraDataBinding;
begin
  BindingGroup1.GetBindingForTarget(Edit2).Converter :=
    TIntToColorConverter.Create();
end;
```

Ao acessar o componente BindingGroup1 temos acessos aos seus diversos métodos. Para recuperar um objeto de Binding que foi criado pelo editor de Binding utilizamos o GetBindingForTarget. Sua função é localizar na lista interna de objetos do componente BindingSource, qual que possui o objeto passado no parâmetro como Target. Ao encontrar, o mesmo é retornado.

Todo objeto de Binding no DSharp possui a propriedade Converter, que pode receber a instância de qualquer classe que implemente TValueConverter. Uma vez passada essa referência, o Framework a utiliza na conversão de valores. Para ter tudo funcionando, simplesmente chamamos ConfiguraDataBinding no evento OnCreate do formulário. Execute novamente o aplicativo e preencha o TEdit adicionado.

Criando um View Model básico

Vamos criar um exemplo que remove do formulário a lógica de negócio envolvida e a centraliza em uma classe View Model. Assim, vamos simular uma tela de envio de e-mails.

Criamos um novo projeto do tipo VCL Forms e lhe damos o nome de Demo2. O formulário criado deve ficar como o da **Figura 9**.

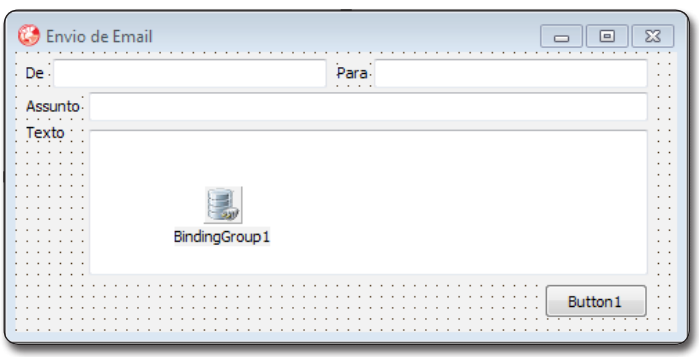


Figura 9. Formulário principal

Não alteramos o nome de nenhum componente para manter a simplicidade, exceto pelo nome do formulário que mudamos para EnvioEmailView e ao salvar todo o projeto, essa unit recebeu o nome de EnvioEmailViewForm.

Nosso View Model será uma classe separada, então adicionamos uma nova unit e salvamos com o nome de EnvioEmailViewModel. O Framework DSharp disponibiliza uma classe base chamada TViewModelBase que para utilizá-la é preciso referenciar em uses a unit: DSharp.PresentationModel.ViewModelBase.

O objetivo do View Model é ser o contexto de dados e comportamento para uma View, dessa forma teremos propriedades para cada dado a ser exibido. Sobre o comportamento, apenas podemos enviar um e-mail se todos os campos estiverem preenchidos. Falando em termos da IU, o botão será habilitado só quando tudo estiver preenchido. A **Listagem 2** mostra nosso View Model.

Implementamos dois membros que representam essa lógica de negócio citada. A propriedade IsValid é utilizada para indicar à View se tudo está correto para o envio de um e-mail. Essa verificação é feita pela função Validate, que simplesmente retorna True quando está tudo preenchido ou False quando falta algo. Ela é chamada em cada "Set" das propriedades que serão ligadas à View por Binding, como mostra a **Listagem 3**.

Listagem 2. View model

```
type
 TEnvioEmailViewModel = class(TViewModelBase)
 private
  //código removido por questões de espaço
 public
    property Remetente: string read FRemetente write SetRemetente;
    property Destinatario: string read FDestinatario write SetDestinatario;
    property Assunto: string read FAssunto write SetAssunto;
    property Conteudo: string read FConteudo write SetConteudo;
    property IsValid:Boolean read FIsValid write SetIsValid;
    function Validate: Boolean; override;
 end;
```

Listagem 3. Realizando a validação

```
procedure TEnvioEmailViewModel.SetAssunto(const Value: string);
begin
 FAssunto := Value;
 DoPropertyChanged('Assunto');
 IsValid := Validate;
end;
```

Sempre que um valor é digitado na View, o mesmo é passado por Binding à sua propriedade correspondente, o que por sua vez dispara o método Set dessa propriedade. Assim que a mesma recebe o valor, é disparado o evento DoPropertyChanged, passando o nome da propriedade em questão. Isso faz com o que o Framework de Binding atualize todas as ligações relacionadas a essa propriedade. E como o objetivo é ir validando o estado geral a cada digitação, nós setamos o valor da propriedade IsValid com o resultado da função Validate. Este é o View Model.

Na View, que é nosso formulário, adicionamos em uses uma referência à nossa unit EnvioEmailViewModel e criamos um campo privado do tipo do View Model: "FDataContext: TEnvioEmailViewModel"; Esse campo será utilizado para realizarmos as operações de Binding. Essas operações não serão criadas de forma manual, mas através de código. Adicionamos uma procedure ao formulário que centraliza essa rotina, como mostra a **Listagem 4**.

Listagem 4. Construindo os bindings

```
procedure TEnvioEmailView.ConfiguraBinding;
begin
 BindingGroup1.AddBinding(FDataContext,'Remetente',Edit1,'Text');
 BindingGroup1.AddBinding(FDataContext,'Destinatario',Edit2,'Text');
 BindingGroup1.AddBinding(FDataContext,'Assunto',Edit3,'Text');
 BindingGroup1.AddBinding(FDataContext,'Conteudo',Memo1,'Text');
 BindingGroup1.AddBinding(FDataContext,'IsValid',Button1,'Enabled');
end;
```

O componente TBindingGroup oferece um método simples para criação de Binding, o AddBinding. Nele passamos uma instância de objeto que será o Source, seguida de uma String que é o nome de uma propriedade desse mesmo Source. Deste modo informamos quem será o Target e qual propriedade estará ligada à propriedade do Source. Veja que a propriedade IsValid foi ligada à propriedade Enabled do botão, isso irá controlar o botão automaticamente quando preenchemos os campos. Antes de executarmos os exemplo, basta adicionar no evento OnCreate do formulário a criação do campo FViewModel e a chamada do método ConfiguraBinding. Feito isso, execute e veja que o botão só é ativo quando tudo está preenchido.

Melhorando a validação

É uma boa prática informar ao usuário o que está inválido. A classe TViewModelBase expõe um método que pode ser sobrecarregado em nosso View Model para aplicar nossas regras de validação e retornar uma mensagem. Como utilizamos Binding, essa mensagem pode ser ligada aos Hints dos controles visuais, ou até mesmo algo mais visual pode ser criado, como a mudança de cor do próprio campo.

Para alcançar esse objetivo, a primeira coisa é sobrecarregar no View Model o método "GetItem(const AName: string): string". Isso é visto na **Listagem 5**.

Listagem 5. Retornando uma mensagem de erro

```
01 function TEnvioEmailViewModel.GetItem(const AName: string): string;
02 const
03  Propriedades: array[0..3] of string = (
04   'Remetente',
05   'Destinatario',
06   'Assunto',
07   'Conteudo');
08 begin
09  case IndexText(AName, Propriedades) of
10   0:
11   begin
12    if not IsValidEmail(Remetente) then
13     Result := 'Remetente inválido';
14   end;
15   1:begin
16    if not IsValidEmail(Destinatario) then
17     Result := 'Destinatário inválido';
18   end;
19   2:begin
20    if Trim(Assunto)='' then
21     Result := 'Assunto não pode ser vazio';
22   end;
23   3:begin
24    if Trim(Conteudo)='' then
25     Result := 'O email precisa de um conteúdo';
26   end;
27  end;
28 end;
```

Esse método é chamado internamente pelo Framework DSharp sempre que uma propriedade envolvida em Binding recebe um valor. Esse passa no parâmetro o nome da propriedade que está em uso no momento. Para não criar vários comandos if de forma encadeada, foi criado um Array (linha 3) com os nomes das propriedades disponíveis e então, através da função IndexText conseguimos localizar a posição dentro do Array do texto que está vindo como parâmetro. Essa função está disponível na unit StrUtils e foi adicionada na seção implementation. Pode ser observado também o uso de uma função chamada IsValid Email, na qual foi implementada para verificar se a String do valor da propriedade confere com um possível e-mail, ou seja, se ela contém um "@" ou um ".com" pelo menos. Inclusive a função Validate criada anteriormente foi alterada para fazer uso dessa função. Sua implementação completa pode ser vista no download do código fonte. Essa é a única alteração realizada no View Model.

Para que o mecanismo de Binding possa detectar a validação em uso e mostrar as mensagens de erro, é preciso configurar os Bindings. A **Listagem 6** mostra como o método ConfiguraBinding foi ajustado.

Na linha 3 foi definida uma variável do tipo TBinding, esse é o tipo que o Framework cria quando adicionamos um novo Binding, seja via código ou pelo Designer. Como criamos os Bindings por código, ele é utilizado para configurarmos opções que só seriam possíveis pelo próprio editor.

Na linha 5 criamos o primeiro Binding. O método AddBinding retorna o objeto TBinding recém criado e sua referência fica

contida na variável "binding". Com isso acessamos a propriedade ValidadesOnDataErrors, setando True. É isso que faz com que o Binding converse com a validação que implementamos no View Model. Isso é bom, pois automaticamente os objetos serão validados. Porém, existe um efeito colateral. Não é possível utilizar o modo de atualização OnPropertyChanged. É necessário alterar para algo mais simples, neste caso o modo utLostFocus. Esse modo apenas atualiza o Source quando o Garget perde o foco. Ou seja, se o Binding é realizado com um TEdit, o Source ligado a esse só é atualizado quando saímos desse TEdit. O valor utLostFocus está contido na unit DSharp.Bindings.Notifications que foi adicionada em uses da seção implementation.

Listagem 6. Configurando os bindings

```
01 procedure TEnvioEmailView.ConfiguraBinding;
02 var
03   binding: TBinding;
04 begin
05   binding := BindingGroup1.AddBinding(FDataContext,'Remetente',Edit1,'Text');
06   binding.ValidatesOnDataErrors := True;
07   binding.SourceUpdateTrigger := utLostFocus;
08   BindingGroup1.AddBinding(binding, 'ValidationErrors[0].ErrorContent',
     Edit1, 'Hint');
09
10   binding := BindingGroup1.AddBinding(FDataContext,'Destinatario',Edit2,
     'Text');
11   binding.ValidatesOnDataErrors := True;
12   binding.SourceUpdateTrigger := utLostFocus;
13   BindingGroup1.AddBinding(binding, 'ValidationErrors[0].ErrorContent',
     Edit2, 'Hint');
14
15   binding := BindingGroup1.AddBinding(FDataContext,'Assunto',Edit3,'Text');
     binding.ValidatesOnDataErrors := True;
16   binding.SourceUpdateTrigger := utLostFocus;
17   BindingGroup1.AddBinding(binding, 'ValidationErrors[0].ErrorContent', Edit3,
     'Hint');
18
19  binding := BindingGroup1.AddBinding(FDataContext,'Conteudo',Memo1,'Text');
20   binding.ValidatesOnDataErrors := True;
21   binding.SourceUpdateTrigger := utLostFocus;
22   BindingGroup1.AddBinding(binding, 'ValidationErrors[0].ErrorContent',
     Memo1, 'Hint');
23
24   BindingGroup1.AddBinding(FDataContext, 'IsValid',Button1,'Enabled');
25 end;
```

Com isso finalizado é possível então realizar um Binding da mensagem de erro a um controle. Isso é visto na linha 8. Observe que agora o Source do Binding não é mais nosso objeto de contexto, mas sim o Binding criado anteriormente. A classe TBinding possui uma propriedade ValidationErros que armazena os erros existentes. Assim, acessamos a propriedade Content desse objeto para exibição da mensagem. Essa mensagem está ligada à propriedade Hint do TEdit. Isso faz com que cada erro apareça em seu respectivo controle.

O mesmo é repetido para as outras propriedades, como pode ser visto. Para que o hint dos controles apareça é preciso habilitar a propriedade ShowHint do formulário. Execute e veja as mensagens de erro aparecendo.

Vantagens

Uma das maiores vantagens em separar de forma correta as camadas lógicas é a testabilidade. É possível testarmos de forma automática o comportamento de uma interface, sem ter que executar essa interface. Vamos demonstrar isso usando os recursos do próprio IDE. Desde a versão 2005 o Delphi traz integrado ao IDE o DUnit. Este Framework implementa tudo o que precisamos para começar a escrever classes de testes unitários sem precisar configurar nada, pois todo o necessário já está implementado pelo Framework. De uma forma simples podemos dizer que um teste unitário é uma classe que aplica um teste em outra classe. Vamos criar o projeto de teste.

Ao clicar com o botão direito do mouse no ProjectGroup da demonstração, temos a opção Add New Project. Ao escolher essa, uma nova janela surge, onde podemos escolher o tipo de projeto de teste, como mostra a **Figura 10**.

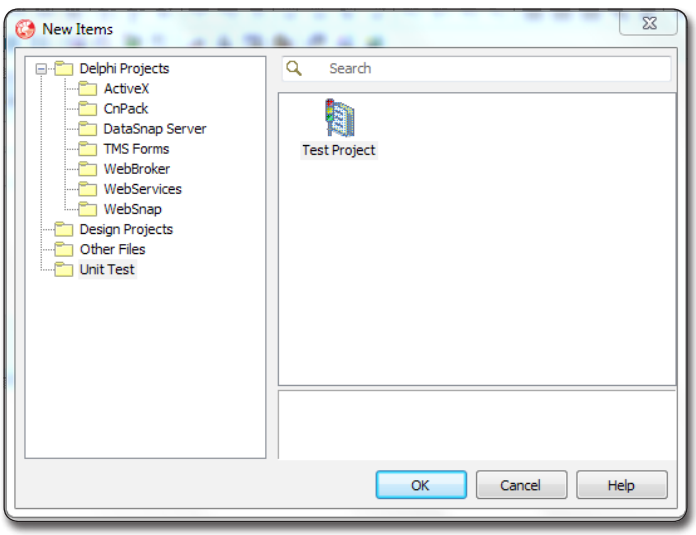


Figura 10. Novo projeto de testes

Ao confirmamos, um assistente de criação é executado, basta aceitar os valores padrão que teremos um projeto configurado para execução de testes. Como este irá testar a classe View Model do projeto anterior, é necessário incluir no "Search Path" onde localizar a classe View Model. Acessando as opções do projeto de teste encontramos a propriedade "Search Path". Nela já existe o caminho de localização do código fonte do Framework DUnit, então adicionamos apenas o caminho para o projeto a ser testado.

Adicionamos também a esse projeto criado uma nova unit chamada TestEnvioEmailViewModel. Nela é preciso adicionar à seção uses a unit TestFramework, é nela que estão declaradas funções e tipos necessários para criação de teste. Criamos também uma classe chamada de TTestViewModel e que herda de TTestCase que é uma classe base do Framework DUnit, representando uma coleção de rotinas de teste, ou seja, é um caso de teste. A essa classe criamos então o primeiro teste. Como testes são classes que testam outras classes, esse primeiro teste nada mais que é uma procedure. No caso, foi adicionada uma procedure chamada TesteRemetenteInvalidoBloqueiaEmail.

Ela deve verificar se, quando temos apenas uma propriedade inválida no View Model, o mesmo permanecerá inválido. A **Listagem 7** mostra a classe e sua implementação.

Listagem 7. Classe de teste

```
01 type
02  TTesteViewModel = class (TTestCase)
03  published
04    procedure TesteRemetenteInvalidoBloqueiaEmail;
05  end;
06
07 implementation
08
09 { TTesteViewModel }
10
11 uses EnvioEmailViewModel;
12
13 procedure TTesteViewModel.TesteRemetenteInvalidoBloqueiaEmail;
14 var
15  viewModel: TEnvioEmailViewModel;
16 begin
17  viewModel := TEnvioEmailViewModel.Create;
18  viewModel.Remetente := 'emailnaoformatado';
19  viewModel.Destinatario := 'email@ok.com.br';
20  viewModel.Assunto:= 'assunto ok';
21  viewModel.Conteudo := 'conteudo ok';
22  CheckFalse(viewModel.IsValid);
23 end;
```

A procedure de teste está localizada na linha 4 e como pode ser visto está posicionada no bloco published. A seção published torna públicos os tipos e métodos nela definidos e ainda registra essas informações para serem utilizadas pela RTTI. Ter os métodos de teste na seção published é uma exigência técnica do DUnit, caso contrário ele não encontrará os testes. Isso acontece porque o framework utiliza a RTTI para identificar através de reflexão os métodos de teste e assim, gerar a tela de execução destes (**Figura 11**)

Nota-se que foi adicionada (linha 11) uma referência à unit do View Model a ser testado, isso é necessário, pois vamos simular o uso do mesmo por uma View, ou seja, vamos instanciar o View Model e setar valores de acordo com o teste.

No caso, como mostra o teste da linha 13, um View Model foi criado e apenas a propriedade Remetente foi preenchida de forma indevida.

O teste em si mesmo acontece na linha 22 com o uso do método CheckFalse. Esse método, pertencente ao Framework DUnit e avalia se o valor passado é false. Se o mesmo for, o teste passou de forma positiva, caso contrário o teste está falho. Existem vários outros métodos disponíveis, como CheckTrue, Check ou Check Equals por exemplo. Se executarmos o projeto de teste nesse momento nada acontecerá. Isso porque é preciso informar ao DUnit que existe um caso de teste. Isso é feito na seção initialization, como mostra o código a seguir:

```
initialization
 RegisterTest(TTesteViewModel.Suite);
```

Nós registramos a existência do caso de teste TTestViewModel. Suite, se houvessem outros, bastaria fazer o mesmo para cada um. Agora ao executar projeto de teste vemos sua execução, como pode ser observado na **Figura 11**.

Como o caso de teste foi registrado, o DUnit cria uma interface para executar tudo o que está registrado. Observe que o método TesteRemetenteInvalidoBloqueiaEmail foi marcado automaticamente. Ao clicar em cada teste é possível escolher qual se deseja executar. Execute o teste clicando no botão "play" de cor verde e veja o resultado (**Figura 12**)

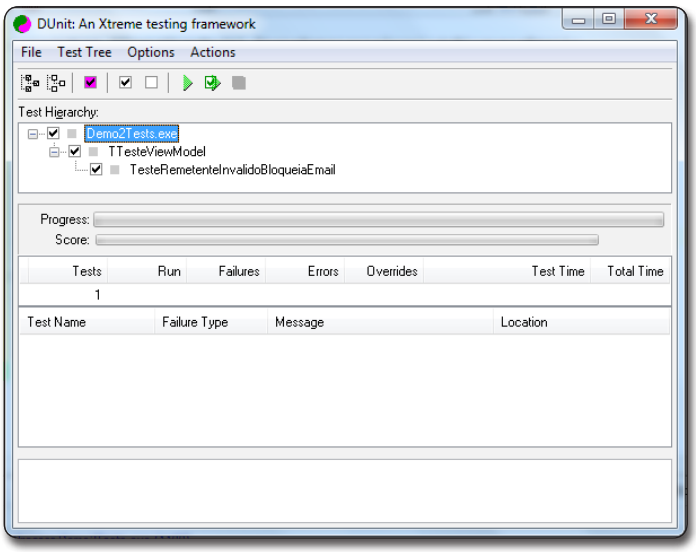


Figura 11. Interface para execução de testes

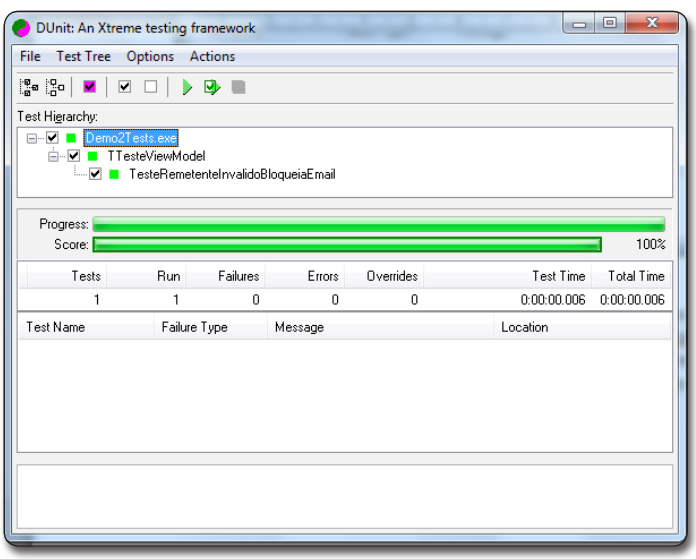


Figura 12. Teste executado com sucesso

O funcionamento da tela está ok, se apenas uma informação está errada, o estado de validação é falso.

Para continuar com os testes, deveríamos codificar pensando nas mais diversas possibilidades. Por exemplo, testar se o View

Model permanece inválido somente com outra propriedade de valor incorreto.

Conclusão

Separar a lógica da tela em uma classe é uma boa prática que deveria ser perseguida por todos. Isso facilita a aplicação de testes unitários, assim seria possível garantir que uma determinada tela está funcionando porque o seu comportamento foi testado.

Neste artigo exploramos somente a casca deste Framework, em artigos futuros podemos aprofundar mais em seus outros conceitos como injeção de dependência e convenção sobre configuração. Um abraço e até mais.

DSharp – Fontes
https://code.google.com/p/delphisorcery/source/checkout

Paulo Quicoli
pauloquicoli@gmail.com
Editor da revista ClubeDelphi e editor técnico .Net Magazine. Formado em processamento de dados pela FATEC-TQ. Atua como analista programador de sistemas e coordenador técnico da equipe de desenvolvimento .NET da Siplan Control-M (www.siplan-controlm.com.br) unidade Jaboticabal, e professor da FATEC, campus Taquaritinga.
Blog: http://quicoli.worpress.com

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

CodeSite no Delphi XE 2

Depuração e criação de logs

O desenvolvimento de aplicações é um processo que envolve diversas etapas que se iniciam no planejamento e seguem até as fases de manutenção e atualização. Contudo, quando um Software começa a receber uma série de recursos e se torna cada vez mais complexo, é comum que erros e outros contratempos apareçam, embora isto ocorra também em aplicações de pequeno porte. Muitas vezes, as falhas são fáceis de serem identificadas e corrigidas, porém, erros difíceis de serem encontrados também fazem parte do cotidiano da equipe de desenvolvimento. Diante deste fator, um recurso extremamente comum que é adotado é a depuração (Debug), que pode estar presente na própria ferramenta de desenvolvimento, através de Softwares de terceiros ou ainda ser efetuada de forma manual, através da geração de arquivos de Log ou mensagens.

O Delphi conta com uma ferramenta de Debug integrada em seu ambiente, que auxilia no processo de análise e identificação de falhas, onde recursos específicos como os Breakpoints podem ser utilizados. Todavia, a identificação de certos tipos de erros pode ser uma tarefa demorada, onde surgem diversas dificuldades que devem ser consideradas. Dentro deste fator, existe a ferramenta CodeSite desenvolvida pela Raize Software e que acompanha o Delphi e as outros IDEs do RadStudio desde as versões mais antigas, oferecendo diversas funcionalidades para o recurso de depuração, permitindo uma análise mais rápida e detalhada do Software para os desenvolvedores e garantindo assim uma melhor qualidade do mesmo. Através dele, o desenvolvedor pode inserir comandos no código fonte da aplicação em questão com o intuito de enviar mensagens para as ferramentas de visualização que compõem o CodeSite e assim, poder analisar o comportamento da mesma de uma forma otimizada e no momento em que as ações são executadas, facilitando assim a descoberta de erros. Este processo pode ser efetuado sem que a execução do programa seja interrompida, uma vez que traz consigo um novo modelo de depuração, dispensando o uso de breakpoints, permitindo que logs possam ser gerados e facilmente consultados para que as prováveis falhas possam ser identificadas. Vale lembrar que o CodeSite também não se limita a estruturas de dados simples, registrando além dos processos e falhas, o momento em que as mesmas ocorrem. O sistema de registros do CodeSite é baseado praticamente em três componentes principais, sendo eles as classes de Log (Logging Classes), o Dispatcher (despachante) e os visualizadores (Viewers). Para o processo de log a classe

Resumo DevMan

De que se trata o artigo:

Este artigo demonstra uma importante ferramenta para auxiliar na depuração das aplicações desenvolvidas no Delphi XE 2, o CodeSite, abordando suas principais características, componentes e formas de empregá-la no dia-a-dia, com o intuito de ajudar na identificação e solução de possíveis falhas de execução, bem como registrar tudo o que ocorre no Software através de arquivos de Log.

Em que situação o tema é útil:

Agilizar no processo de identificação de erros presentes na aplicação que podem ser difíceis de ser encontrados por meio dos métodos tradicionais de depuração, empregando formas avançadas para este processo, visando assim uma maior produtividade para o desenvolvedor e qualidade para o Software final.

CodeSite:

No ambiente de desenvolvimento de Softwares, os indesejáveis "erros" são praticamente inevitáveis, principalmente em casos onde complexas rotinas devem ser empregadas. Evidentemente, problemas simples podem ser resolvidos de uma forma rápida mesmo sem uma avançada ferramenta de auxílio, apenas utilizando um depurador básico e o recurso de breakpoints, revisando o trecho que apresenta o problema, mas nem sempre este processo é assim. Podem surgir falhas complexas ou erros pouco detalhados, fazendo com que o desenvolvedor tenha que revirar a aplicação em busca de uma reposta, perdendo muito tempo que poderia ser empregado em outras tarefas. Diante deste fator, existe uma ferramenta que acompanha a suíte de aplicativos do RAD Studio, o CodeSite. Esta ferramenta incorpora a implementação de novas técnicas para auxiliar na identificação das possíveis falhas, através de métodos e visualizadores que permitem um monitoramento dos processos em tempo real ou ainda, através da gravação de arquivos de Log. Neste artigo serão apresentadas formas de gerar informações sobre os processos internos do Software, facilitando assim a identificação de erros, primeiramente adotando um processo manual, através da simples gravação de um arquivo txt e posteriormente, de uma maneira mais ampla e produtiva, com a utilização dos recursos da ferramenta CodeSite.

principal que pode ser utilizada no Delphi é a TCodeSiteLogger, que por sua vez possui uma diversidade de funções, sendo o método Send um dos mais utilizados, já que o mesmo permite o envio de diversas mensagens para a ferramenta. Este método também é sobrecarregado (overload), suportando diversos tipos para o envio das mensagens. Outra classe que pode ser empregada na utilização da ferramenta CodeSite é a TCodeSiteManager. Ela possui várias propriedades que podem ser definidas para as instâncias da classe TCodeSiteLogger. A próxima ferramenta, o Dispatcher, assim como seu nome sugere, é um processo que roda em segundo plano na qual tem o objetivo de transferir as mensagens enviadas pelos métodos da classe TCodeSiteLogger para um destino final, que pode ser uma aplicação rodando em paralelo ou um arquivo presente no disco rígido, localmente ou remotamente através de uma rede. Contudo, apesar das mensagens serem enviadas e se tornarem uma tarefa a mais para aplicação, o desempenho da mesma é mantido. Também existe uma aplicação externa que permite o gerenciamento de algumas funcionalidades da ferramenta Dispatcher, o CodeSite Controller. Já os visualizadores são as outras ferramentas presentes no CodeSite para que as informações geradas possam ser visualizadas e analisadas em tempo real, através da ferramenta CodeSite Live Viewer, ou posteriormente, por meio do carregamento de um arquivo de log, através da ferramenta CodeSite File Viewer, onde diversos recursos estão presentes para que a informação possa ser filtrada. Vale a pena salientar também que a extensão adotada para os arquivos de log do CodeSite é a csl. Outro ponto interessante é que as ferramentas podem ser rapidamente executadas através do Delphi através do menu Tools>CodeSite.

Com relação à versão que acompanha o Delphi XE 2, a 5.0 Express, os recursos foram aprimorados com relação a versão anterior, tais como a implantação do processo através da classe TCodeSiteLogger e seus métodos, o registro das informações, a usabilidade e a produtividade. Também é importante ressaltar que a Unit CodeSiteLogging deve ser incluída na seção Uses para que as classes da ferramenta possam ser utilizadas.

Um recurso interessante, porém, não está disponível na versão que acompanha o Delphi XE 2 é o Method Tracer, que por sua vez permite que todos os métodos e classes presentes em uma Unit possam ser visualizados em uma janela em forma de árvore, auxiliando o desenvolvedor na definição de quais métodos devem ser rastreados, facilitando a adição dos métodos EnterMethod (Entrada em um método) e ExitMethod (Saída do método).

Tutorial

Logs na prática

Para dar início à geração de logs em arquivos na prática, será criado um novo projeto através do menu File>New>VCL Forms Application. O mesmo foi salvo com o nome de Log. Para este exemplo, em um primeiro momento será gerado um log de forma manual e posteriormente, utilizando a ferramenta CodeSite. Sendo assim, foi criado um banco de dados Firebird e nele adicionada uma simples tabela de clientes para que erros fossem simulados e tais processos sejam colocados em prática. A **Listagem 1** apresenta o código SQL da tabela de clientes.

Listagem 1. Código SQL da tabela Clientes

```
CREATE TABLE CLIENTES (
   ID INTEGER NOT NULL,
   NOME VARCHAR(50),
   ENDERECO VARCHAR(50),
   CONSTRAINT PKY_CLIENTE PRIMARY KEY(ID)
);
```

Com a criação da tabela, foram adicionados ao formulário principal do projeto, um objeto do tipo SQLConnection, 2 Labels, 2 Edits e 2 SpeedButtons. O objeto SQLConnection1, responsável por efetuar a conexão com o banco de dados Firebird, teve suas propriedades definidas de acordo com o trecho a seguir:

```
ConnectionName: FBConnetion
LoginPrompt: False
Params.Database: [Endereço do banco]
Connected: True
```

Como pode ser observado, a propriedade ConnectionName indica o banco de dados (FBConnection), LoginPrompt definida como falso tem a função de não exibir a janela de Login no banco de dados. Params.Database especifica o endereço do arquivo de dados e por último, a propriedade Connected definida como verdadeiro, permite a conexão automática com o banco.

A próxima etapa do projeto é definir o método que será responsável por gerar as informações para o arquivo de log manualmente. Vale à pena lembrar que um log geralmente é um arquivo de texto puro que recebe os dados com a data, hora e a ação que foi efetuada. Entretanto em algumas aplicações as mensagens de log podem ser armazenadas em tabelas do banco de dados, e indo um pouco mais além da função de monitorar os processos do sistema, podem também ser utilizadas para registrar as ações dos usuários, atuando de forma semelhante a um histórico. Sendo assim, a classe TextFile do Delphi, responsável por prover as funcionalidades para arquivos de texto, pode ser empregada para a criação e manipulação de um arquivo de log simples. A **Listagem 2** apresenta o procedimento que foi elaborado para a geração e gravação manual das informações do arquivo de log.

Neste trecho de código, logo após a definição das variáveis, o endereço físico para a gravação do arquivo é definido contendo a raiz do disco rígido e seu devido nome. Em seguida, a função AssignFile é utilizada para efetuar a associação entre o arquivo externo e sua devida variável. De acordo com a função FilesExists, é verificado se o arquivo já existe no endereço especificado, e se o retorno é verdadeiro. Caso esta afirmação seja verdadeira, ele

é aberto para que a informação possa ser adicionada através do método Append, caso contrário, ele é criado através do método ReWrite. Uma vez que o arquivo está preparado, os dados são gravados através de comandos Writeln, onde a data e a hora atuais são obtidas com a utilização da função Now e formatadas corretamente de acordo com FormatDateTime, seguidas pela mensagem em String que foi passada como parâmetro. Para concluir, a associação entre a variável e o arquivo externo é encerrada com o emprego do método CloseFile, que como o próprio nome sugere, encerra a abertura do arquivo.

Listagem 2. Procedimento para geração manual de um arquivo de log

```
procedure TFormLog.GerarLog(Mensagem: string);
var
  ArquivoLog: TextFile;
  Endereco: string;
begin
  Endereco := 'C:\LogSitema.txt';
  AssignFile(ArquivoLog, Endereco);
  if FileExists(Endereco) then
    Append(ArquivoLog)
  else
    ReWrite(ArquivoLog);
  Writeln(ArquivoLog, FormatDateTime('dd/mm/yyyy hh:mm:ss', Now) + ': '
   + Mensagem);
  CloseFile(ArquivoLog);
end;
```

Com a função responsável por gerar o log manualmente definida, a próxima etapa é criar o código que receberá a monitoração, que neste exemplo foi o evento OnClick para o primeiro botão, onde um comando SQL baseado em uma instrução de INSERT foi atribuído de forma errada propositalmente para que o registro do log fosse realizado. Na geração de logs manuais, uma estrutura muito útil que pode ser empregada para o tratamento das exceções e evidentemente, seus registros, é o bloco de tratamento de exceções "try/except", já que o mesmo permite determinar um trecho de código que deve ser tratado caso ocorra alguma falha. Outro ponto que deve ser considerado é com relação aos locais onde a função será chamada, para que as informações desejadas ou de maior risco possam ser capturadas. A **Listagem 3** apresenta o código do evento OnClick para o botão btGravarManual.

Neste trecho de código é possível notar que a classe TSQLQuery foi empregada para a execução das instruções SQL de forma manual para o Framework dbExpress, sendo assim instanciada para o objeto Qry. Em seguida, o procedimento GerarLog é chamado para registrar que o clique no botão foi realizado. Posteriormente, o objeto Qry tem sua propriedade SQLConnection definida para indicar qual a conexão que deve ser empregada na execução da mesma. Na propriedade SQL.Text, utilizada para a definição da instrução SQL, é atribuído um SQL com uma falha ("INSER" no lugar de "INSERT") para que uma exceção seja propositalmente gerada, uma vez que este é um erro muito comum de ocorrer. Neste momento também é adicionada uma nova linha para o log, contendo as informações do comando SQL que foi fornecido.

Listagem 3. Código do envento OnClick para a geração do Log manual

```
procedure TFormLog.btGravarManualClick(Sender: TObject);
var
  Qry: TSQLQuery;
begin
  Qry := TSQLQuery.Create(nil);
  GerarLog('Clicou no botão btGravarManual');
  try
    with Qry do
    begin
      SQLConnection := SQLConnection1;
      SQL.Text := ' INSER INTO CLIENTE '+
            ' VALUES (1,'+
            QuotedStr(edNome.Text) +','+
            QuotedStr(edEndereco.Text) +');';
      GerarLog('Criou o SQL: '+ Copy(SQL.Text, 1, Length(SQL.Text) -1));
      ExecSQL;
      Close;
    end;
  except
    on e:Exception do
      GerarLog('Erro: '+ e.Message +' : '+ e.ClassName);
  end;
  FreeAndNil(Qry);
end;
```

Posteriormente, a Query é executada através do método Exec SQL. Para concluir, sempre que uma exceção for gerada, com o auxílio da estrutura try/except, o procedimento GerarLog é chamado novamente para gravar a mensagem de erro retornada (contida em e.Message) com base na classe Excpetion, finalizando assim o o processo com a liberação do objeto Qry da memória.

Como pode ser observado, a geração de arquivos de log manualmente é um processo que permite um bom monitoramento do que é realizado e que pode ser melhorado para obter informações mais específicas. Também vale ressaltar que em algumas ferramentas de desenvolvimento, o desenvolvedor pode enviar mensagens de log diretamente para um console ao invés de um arquivo. Todavia, quando falhas mais complexas ocorrem, uma ferramenta mais específica deve ser empregada para que os esforços gastos neste processo sejam eliminados, como é o caso do CodeSite.

Logs com o CodeSite

O mesmo projeto criado anteriormente será adotado, onde a Unit CodeSiteLogging foi incluída na seção Uses do formulário. Para fazer um rápido teste e ver o funcionamento da ferramenta, o código a seguir pode ser atribuído no evento OnClick do segundo botão inserido (btGravarCodeSite):

```
CodeSite.Send('Clicou no botão btGravarCodeSite');
```

Ao executar a aplicação e clicar no botão btGravarCodeSite, a ferramenta CodeSite Live Viewer é carregada automaticamente e já exibe a mensagem que foi informada para o método Send. A **Figura 1** apresenta a janela da ferramenta CodeSite Live Viewer com a mensagem retornada.

A utilização da ferramenta CodeSite é bem simples e automatizada para a depuração do código fonte. Para dar continuidade ao projeto, o código do evento OnClick do botão btGravarCodeSite receberá novas funcionalidades para o registro das ações que ocorrem no momento em que a gravação das informações forem efetuadas através de uma Query. A **Listagem 4** apresenta o novo código atribuído ao botão btGravarCodeSite.

Uma TSQLQuery também é empregada para que um comando SQL possa ser utilizado. Após a criação de uma instância da mesma, o comando CodeSite.EnterMethod é utilizado para que uma mensagem seja enviada à ferramenta do CodeSite em utilização, indicando o início de um bloco de códigos para um método, neste caso o evento OnClick do btGravarCodeSite. O método CodeSite.Send é utilizado novamente para o envio de mais informações, tais como os valores presentes nos objetos instanciados edNome e edEndereco. Também é enviada a mensagem contendo o comando da Query (SQL.Text) com a mesma falha proposital apresentada anteriormente (instrução "INSER"). O CodeSite também possui um método específico para o envio de mensagens à partir de uma exceção, o SendException, que é empregado neste trecho. Para concluir, é enviada a mensagem de encerramento do bloco referente ao método OnClick através de ExitMethod e o objeto Qry (TSQLQuery) é liberado da memória através de FreeAndNil.

A **Figura 2** apresenta a janela do CodeSite Live Viewer com o resultado da execução do evento OnClick, apresentado anteriormente, para o botão btGravarCodeSite.

Todas as mensagens enviadas são automaticamente exibidas pela ferramenta CodeSite Live Viewer, apresentando também a exceção que foi gerada para o try/except, já que nenhuma forma de gerenciamento para as mensagens foi especificada até o momento.

Contudo, como mencionado, o CodeSite disponibiliza outras formas para a depuração das aplicações. A próxima que será empregada é a utilização de arquivos de Log para o armazenamento das ocorrências. Para tanto, foi adicionado mais

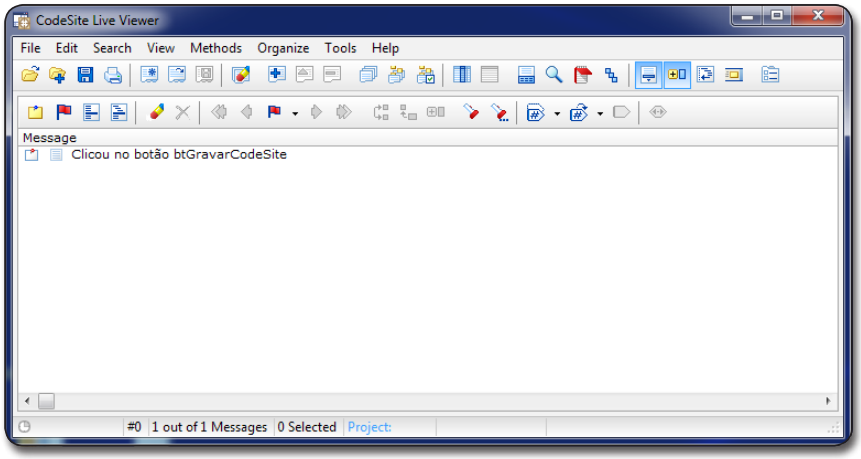


Figura 1. Janela do CodeSite Live Viewer

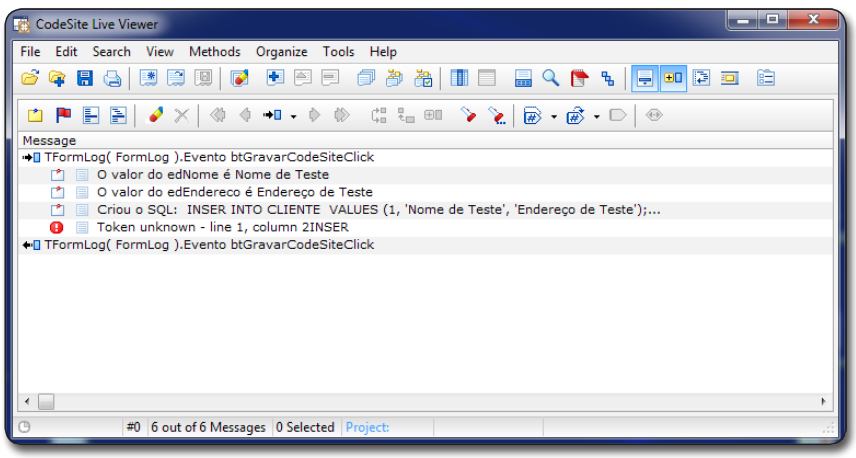


Figura 2. Janela do CodeSite Live Viewer para o botão btGravarCodeSite

Listagem 4. Código do envento OnClick para o botão btGravarCodeSite

```
procedure TFormLog.btGravarCodeSiteClick(Sender: TObject);
var
  Qry: TSQLQuery;
begin
  Qry := TSQLQuery.Create(nil);
  CodeSite.EnterMethod(Self,'Evento btGravarCodeSiteClick');
  CodeSite.Send('O valor do edNome é '+ edNome.Text);
  CodeSite.Send('O valor do edEndereco é '+ edEndereco.Text);
  try
    with Qry do
    begin
      SQLConnection := SQLConnection1;
      SQL.Text := ' INSER INTO CLIENTE '+
            ' VALUES (1,'+
            QuotedStr(edNome.Text) +','+
            QuotedStr(edEndereco.Text) +');';
      CodeSite.Send('Criou o SQL: '+ Copy(SQL.Text, 1, Length(SQL.Text) -1));
      ExecSQL;
      Close;
    end;
  except
    on e:Exception do
      CodeSite.SendException(e);
  end;
  CodeSite.ExitMethod(Self,'Evento btGravarCodeSiteClick');
  FreeAndNil(Qry);
end;
```

Classe/Método	Descrição
TCodeSiteLogger.AddCheckPoint	Incrementa um contador específico para Check points e adiciona uma mensagem de checagem.
TCodeSiteLogger.AddSeparator	Adiciona um separador (linha) para uma organização visual do log.
TCodeSiteLogger.CloseDispatcher	Encerra a ferramenta Dispatcher.
TCodeSiteLogger.ConnectUsingTcp	Utilizado para alterar a conexão com um Dispatcher através do protocolo TCP (Transmission Control Protocol), permitindo assim um monitoramento remoto.
TCodeSiteLogger.SendIf	Envia uma mensagem se uma condição é satisfeita.
TCodeSiteLogger.SendScreenShot	Permite o envio de uma imagem de uma janela específica.
TCodeSiteDestination.HTTP	Permite o envio das mensagens para um servidor de aplicação baseado no protocolo HTTP (HyperText Transfer Protocol).
TCodeSiteDestionation.TCP	Permite o envio das mensagens para uma ferramenta Dispatcher remota através do protocolo TCP.

Tabela 1. Outros métodos da ferramenta CodeSite que podem ser úteis

um botão (TSpeedButton) ao formulário principal. A classe que permite o gerenciamento de como e onde as mensagens enviadas pelos métodos do CodeSite serão gravadas, é a TCodeSiteDestination. Através dela também é possível definir mais de um destino para as mensagens. Outro recurso importante que deve ser empregado neste momento é o CodeSiteManager. Ele tem o objetivo de disponibilizar e gerenciar todas as propriedades para as instâncias da classe TCodeSiteLogger, sendo então utilizado em conjunto com as definições da instância de TCodeSiteDestination. A **Listagem 5** apresenta o código do evento OnClick do botão btGravarCodeSiteLog para a geração de um arquivo de log diretamente no disco.

Listagem 5. Código do envento OnClick para o botão btGravarCodeSiteLog

```
procedure TFormLog.btGravarCodeSiteLogClick(Sender: TObject);
var
  Qry: TSQLQuery;
  Destino: TCodeSiteDestination;
begin
  Qry := TSQLQuery.Create(nil);
  Destino := TCodeSiteDestination.Create(nil);
  with Destino.LogFile do
  begin
    Active   := True;
    FilePath := 'C:\';
    FileName := 'LogSistema.csl';
    MaxSize  := 100;
    MaxParts := 20;
  end;
  Destino.Viewer.Active := False;
  CodeSiteManager.DefaultDestination := Destino;
  //Mesmo código da Listagem 4 ...
  FreeAndNil(Destino);
end;
```

Neste trecho de código é possível notar que uma instância da classe TCodeSiteDestination é criada e atribuída ao objeto Destino após a devida criação de uma instância da classe TSQLQuery. Em seguida, são definidas diversas propriedades para o objeto Destino, sendo elas a Active (True), FilePath, para a definição do diretório onde será gravado o arquivo de Log, FileName para o nome do arquivo, MaxSize para o tamanho máximo do mesmo, e MaxParts, para o número máximo de arquivos. Também é desativada a visualização através do Live Viewer com a subpropriedade Viewer.Active. Posteriormente, o destino padrão (DefaultDestination) do CodeSiteManager é definido com as configurações do objeto Destino. Neste momento, o mesmo código apresentado na **Listagem 4** pode ser empregado para a geração do arquivo de Log. Para concluir, o objeto Destino é liberado da memória através do procedimento FreeAndNil.

Outros recursos da ferramenta CodeSite

Além dos recursos que foram apresentados para auxiliar na depuração de uma simples aplicação, o CodeSite também pode ser empregado para a análise de Threads, onde o método SetCurrentThreadName, presente no objeto CodeSiteManager, pode ser empregado para a identificação da Thread em questão. Em conjunto com o mesmo, os métodos apresentados anteriormente também podem ser empregados para o registro das ocorrências de acordo com a necessidade do projeto.

Se por algum motivo o desenvolvedor necessitar enviar dados personalizados para o CodeSite, ele pode criar classes que implementem a interface ICodeSiteCustomData, classes que derivem de TCodeSiteFormatter ou ainda classes que possuam informações de tipo em tempo de execução usando RTTI (Run Time Type Information), que por sua vez podem ser declaradas com as diretivas de compilação {$M+} e {$M-} , ou simplesmente derivadas da classe TPersistent.

O CodeSite também conta com uma diversidade de métodos que podem ser empregados e que não foram apresentados neste artigo mas que também são úteis. A **Tabela 1** demonstra alguns dos outros métodos que podem ser empregados.

Conclusão

No processo de desenvolvimento de um Software, erros de execução são uma realidade que fazem parte da rotina do desenvolvedor. No Delphi XE 2, assim como em outras versões, o Debugger integrado em conjunto com os Breakpoints, normalmente é o processo mais empregado para ajudar na solução dos erros do sistema, entretanto, podem surgir ocorrências inesperadas muito mais complexas. Se desejar, o desenvolvedor pode

criar rotinas e sobrecarregá-las para gerar Logs manuais e assim poder analisar os erros de uma forma mais específica, mas mesmo assim, isto pode consumir tempo e não contribuir de acordo com o esperado na identificação das falhas. Diante deste fator, é extremamente importante contar com recursos que auxiliem na identificação das falhas em tempo real ou através do registro das informações dos processos realizados, e uma ferramenta que ajuda neste contexto é o CodeSite. Ele é constituído por diversos recursos que oferecem uma maneira mais abrangente para a depuração do código fonte, permitindo que o desenvolvedor encontre os erros da aplicação de uma forma mais rápida e simples, aumentando assim a produtividade.

Quanto ao funcionamento, o CodeSite pode ser empregado para a análise da aplicação através das suas classes e ferramentas, onde o desenvolvedor pode inserir métodos internos que enviam mensagens para o CodeSite Live Viewer ou diretamente para arquivos no disco rígido, contando com diversos recursos para a personalização deste processo.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Raize Software e CodeSite
http://www.raize.com/

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

InterBase XE – Parte 1

Conheça este SGBD de alta performance, leve e escalável da Embarcadero

Este artigo faz parte de um curso

O InterBase é o SGBD atualmente desenvolvido, mantido e comercializado pela Embarcadero, a mesma fabricante do Delphi. Como se sabe SGDB é o acrônimo de Sistema de Gerenciamento de Banco de dados. Esta denominação é uma referência traduzida do termo em inglês DBMS (Database Management System). No entanto, no caso do InterBase, comumente é utilizada a sigla RDBMS (Relational Database Management System), algo como Sistema de Gerenciamento de Banco de Dados Relacional, especificando assim sua característica. Adicionalmente, ele é um sistema bastante convencional e totalmente compatível com a SQL-92 (ver **nota do DevMan 1**).

Partindo de uma comparação inicial com outros SGBDs, o grande atrativo do InterBase fica por conta de seu tamanho enxuto, alto desempenho e quase zero de requisitos de administração. Em relação a este último, significa dizer que o próprio desenvolvedor é capaz de administrá-lo (como já acontece em muito dos casos), extinguindo a necessidade de um profissional especializado, tal como um administrador de banco de dados (DBA). Este cenário de dependência é o que ocorre corriqueiramente com SGBDs do porte de Microsoft SQL Server, Oracle e DB2. No mais, a conectividade do InterBase se estende a vários sistemas operacionais diferentes, mais precisamente Windows, MacOS X, Linux e Solaris.

Um pouco de história

Podemos pontuar a década de 90 como o início da história do InterBase, no momento em que a Borland estabelecia a aquisição de uma empresa até então proprietária do produto, e que culminou com a inclusão do InterBase como parte do negócio. Na década seguinte, em meados de 2000, a Borland traçou uma estratégia para um desmembramento que tinha como objetivo determinar uma empresa específica para tomar a frente do produto.

Resumo DevMan

De que se trata o artigo:

O InterBase é o SGBD da mesma fabricante atual do Delphi, a Embarcadero, mas que esteve fora de pauta por um longo tempo dentro da comunidade, apesar de estar presente no mercado desde seu lançamento inicial há mais de uma década atrás. Por motivos principalmente relacionados ao crescimento da própria comunidade nos últimos tempos, novos desenvolvedores talvez não tiveram contato com o produto. Sendo assim, o presente artigo remonta a história e faz uma apresentação do InterBase, dando um foco maior na versão mais recente do produto, ao mesmo tempo em que é exposto um pouco de suas principais características onde muitas delas são provindas de seus releases anteriores.

Em que situação o tema é útil:

Demonstrar as qualidades do banco de dados da Embarcadero, permitindo assim, acrescentar ainda mais opções para uma importante etapa do desenvolvimento, a escolha da solução de armazenamento.

InterBase XE – Parte 1:

Historicamente falando, como é de conhecimento de muitos, Delphi e InterBase sempre estiveram relacionados. Isso se justifica pelo fato de ambos sempre serem geridos por uma mesma fabricante, o que resulta inclusive em diversos recursos nativos. Exemplo disso é uma paleta de componentes exclusiva (IBX) dentro do IDE de desenvolvimento, bem como um driver dbExpress otimizado. Apesar de estarem ligados, o que se percebe no cenário atual é que o Delphi obteve muito mais sucesso em termos de visibilidade e usabilidade. O InterBase, por si só, apesar de ter ficado à sua sombra, sempre esteve no mercado e, como qualquer produto comercial, foi constantemente melhorado a cada novo lançamento. Dito isto, este artigo tem a intenção de retomar o InterBase em meio à comunidade, mostrando um pouco do estágio atual deste SGBD, fazendo um overview de algumas de suas principais características presentes em sua versão mais recente (XE), sejam elas novas ou integrantes de seu legado. Por se tratar de um assunto extenso, o tema será abordado numa série de dois artigos, onde este primeiro terá como foco a apresentação geral do produto em si, deixando para o segundo uma abordagem mais prática, mas não menos conceitual.

Com o insucesso deste plano, fundamentalmente devido a motivos políticos, o InterBase manteve-se sob propriedade da Borland ao mesmo tempo em que seu código foi exposto ao grande público sob uma variante da Mozilla Public License, se tornando um projeto Open Source. Neste momento o produto encontrava-se na versão 6 e logo tomou fama na comunidade Delphi, sendo largamente adotado por desenvolvedores e empresas na época. Após um breve período de pequenas atualizações, passando inclusive pela versão 6.5 do produto, a Borland anuncia o fim da gestão do código aberto, dando abertura para que outros grupos ou empresas pudessem tomar as rédeas do projeto. Apenas para citar, o exemplo mais clássico disso é o Firebird que, no ano de 2000 foi concebido por um grupo de programadores, a partir do código do InterBase e cujo projeto permanece ativo até os dias atuais. Com o surgimento do Firebird, e seu progressivo amadurecimento, a comunidade de desenvolvedores começa a ver nele uma solução eficaz para a parte de dados de suas aplicações, em vista de sua eficácia e de seu caráter gratuito. Sendo assim o InterBase começa naturalmente a perder espaço, tendo sua presença notada mais em sistemas legados.

Nota do DevMan 1

SQL-92 é a denominação dada à terceira revisão da linguagem de consulta de banco de dados, a SQL (Structured Query Language). Nesta emenda, foram estabelecidos novos recursos que acabaram por se tornar algo trivial nos dias atuais, tais como a adoção de tipos de dados como Date, Time e TimeStamp, o suporte a alterações de esquema através de comando ALTER e DROP, bem como o trabalho com tabelas temporárias.

Seguindo uma ordem cronológica, o produto ainda contou com novos releases nos anos conseguintes. Apenas como informativo, em 2002 a Borland lançava o InterBase 7, que trazia uma melhoria significativa na parte de recursos para servidor, tais como ferramentas de monitoramento e controle. Nos anos seguintes, esta versão se manteve como base, refletindo nos lançamentos 7.1 (2003), 7.5 (2004) e 7.5.1 (2005). Dando um salto para 2006, acontece o anúncio do InterBase 2007, com previsões de melhorias significativas, como backup incremental, caracteres Unicode e novo driver ODBC exclusivo. Nesta mesma época, já com a mudança de Borland para CodeGear, o produto se mantém presente mas sem grandes alardes, deixando de ser pauta na comunidade Delphi. Dois anos depois, já em 2008, a CodeGear é então adquirida pela atual Embarcadero. Sob esta nova proprietária, é apresentado o InterBase 2009, que incluiu novas características que perduram até sua versão mais recente, tal como suporte a criptografia multinível (de banco e colunas). Por fim, no ano de 2010 é lançada a versão XE do produto, com versões para 64-bits para suas partes servidor e cliente. Versão esta que se encontra o produto atualmente, até o momento da escrita desta publicação.

Características

Falando um pouco mais sobre as características do InterBase, podemos estabelecer que ele é um banco de dados rápido. Quando inserido no desenvolvimento Delphi, essa qualidade se torna ainda maior pela presença de componentes de acesso a dados nativos, arquitetados sob medida para a tecnologia, tal como os da paleta InterBase. A **Figura 1** mostra este grupo de componentes sob a perspectiva do Delphi XE2. Ainda se tratando de performance, o SGBD trabalha com multiprocessamento simétrico, suportando múltiplos processadores, incluindo CPU's multi-core. Além dessas, podemos facilmente citar algumas outras de suas principais características técnicas:

- Protocolo de Rede: o InterBase trabalha com o protocolo TCP/IP em todas as suas plataformas suportadas. Especificamente para ambientes onde sua parte cliente e servidor estejam inseridos na plataforma Windows, acrescenta-se o suporte a NetBEUI/named pipes;
- SQL-92: trabalha em conformidade com o padrão ANSI SQL, seja através de suas ferramentas interativas, seja através de aplicações Embarcadero para Desktop (Delphi e C++ Builder);
- Simultaneidade de acesso a banco: provê suporte para que uma única aplicação possa acessar vários bancos de dados distintos ao mesmo tempo, num mesmo servidor;
- Arquitetura multigeracional: o InterBase Server estende, conforme a necessidade, o armazenamento de versões mais obsoletas dos registros, a fim de prover às transações uma visão mais consistente dos dados solicitados;
- Optimistic row-level locking: numa tradução livre para o português, trata-se de bloqueio otimista em nível de linha. Isso significa que o Server, no momento da atualização de um registro no banco por parte do cliente, faz o bloqueio apenas do referido registro de forma individual, ao invés de bloquear toda a página de banco de dados;
- Otimização de consultas ao banco de dados: o desempenho de uma consulta aos registros do banco de dados, invariavelmente se torna uma preocupação primária para o cliente que faz a chamada. Visando contribuir para este processo, o InterBase Server faz uma otimização automática de todas as Querys executadas sobre ele. Caso não seja suficiente, pode-se especificar então um plano de consulta manualmente;
- Alertas: o InterBase provê Event Alerters, que nada mais são que mensagens que são passadas do banco para a aplicação, possibilitando que estas recebam de forma assíncrona notificações sobre alterações na base de dados;
- Views atualizáveis: o InterBase permite Views atualizáveis, cujas alterações irão refletir diretamente nos dados relacionados à medida que ocorrem;
- Gerenciamento explícito de transações: controle total sobre o ciclo de vida completo de uma transação – start, commit e rollback, bem como transações nomeadas.

Evolução das keywords InterBase

Keywords InterBase nada mais são que as palavras-chave presentes em todos os dialetos de banco de dados do produto.

Tais palavras reservadas são aquelas que você irá utilizar durante a escrita de suas instruções SQL, seja ela de ação ou de manipulação de objetos do banco. Dito isto, na medida em que o InterBase evoluía, de sua versão 6 até a atual XE, novas Keywords foram

sendo introduzidas, refletindo seus recursos relacionados que foram sendo introduzidos neste caminho. A fim de elucidar um pouco deste cenário, podemos citar algumas delas. Por exemplo, palavras corriqueiras relacionadas a campos, tipos e valores, tais como BOOLEAN, TIMESTAMP, DAY, MONTH, YEAR, TRUE, FALSE, ROWS e EXTRACT foram surgindo com maior intensidade até a versão 2007 do produto. Adicionalmente, outras palavras relacionadas a funções específicas, tais como NULLIF, PRESERVE, ENCRYPT, DECRYPT e COALESCE ganharam um espaço maior em releases maiores, tal como as versões 7.5, 2009 e XE.

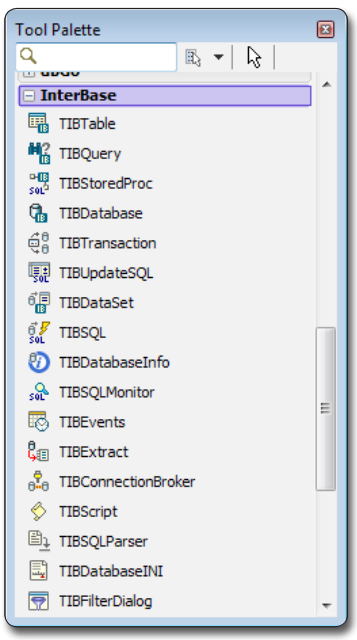


Figura 1. Delphi XE2 - grupo de componentes InterBase

Novidades na versão XE

A família de produtos XE2 da Embarcadero, que contou com produtos como Delphi XE2, RadPHP XE2, Prism XE2, dentre outros, não trouxe consigo uma nova versão do InterBase. Sendo assim, este se encontra, até o momento da escrita deste artigo, na sua versão XE, numericamente expressa pelo número 10. Obviamente, assim como qualquer outro produto de mercado seu Release mais recente contou com novidades que o diferem positivamente de suas versões anteriores. Em suma, os novos recursos podem ser pontuados em fatores como segurança, escalabilidade/desempenho e desenvolvimento de aplicações.

Pela parte de segurança, podemos destacar a melhoria na parte de senhas. Isto, em razão do suporte à criptografia SHA-1 (Secure Hash Algorithm) para sua proteção, e o estabelecimento de um limite estendido de até 32 caracteres de comprimento, a fim de seguir as exigências mais rigorosas de segurança. Apenas fazendo um adentro, o algoritmo SHA-1 é considerado por muitos o sucessor natural do MD5 (Message-Digest Algorithm 5), um algoritmo de hash de 128 bits, muito conhecido dos desenvolvedores.

Em termos de recursos que visam facilitar o desenvolvimento de aplicações, tal quando construímos aplicações Delphi com interação com um banco de dados InterBase, foi introduzido o suporte a SQL dinâmico em Store Procedures, o que garante uma flexibilidade anteriormente inexistente ao desenvolvedor. Finalmente, pelo lado do desempenho, houve uma otimização substancial do manuseio de objetos de banco de dados de tamanho considerável (grande), através de métodos Stream elaborados.

Edições do produto

Como todo produto comercial de médio e grande porte, o InterBase possui diversas edições, em que cada qual se adéqua a um determinado tipo de cenário. A seguir será feita uma explanação de cada uma das edições disponíveis bem como a necessidade que cada uma visa atender. No mais, servirá inclusive para expor ao leitor que o produto possui uma versão gratuita, assim como outros grandes SGBDs conhecidos no mercado, fato este que certamente soará como uma novidade para muitos.

InterBase Server Edition

Server é a edição mais robusta do produto, provendo suporte a múltiplo processamento simétrico, recursos de monitoração do servidor entre outros recursos. Apesar de "leve", o Server Edition é destinado a contextos que lidam com centenas ou milhares de usuários simultâneos. Em vista de necessidades especiais de cenários assim, esta edição contempla alguns recursos específicos, tal como a criptografia de dados, relacionada à segurança das informações. Como exemplo trivial, podemos citar uma empresa corporativa, onde cada uma das diversas estações dos colaboradores (funcionários) roda uma aplicação cliente que faz acesso a uma base dados InterBase localizada num servidor central. Em termos sistêmicos, seriam os casos de sistemas ERP, CRM e etc.

Em termos de volume, o InterBase Server Edition provê opções que vai do número mínimo de cinco até um número ilimitado de usuários, estando disponível para as plataformas Microsoft Windows, Mac OS X, Linux e Solaris.

InterBase Desktop Edition

Desktop é a edição do servidor de banco de dados InterBase projetado exclusivamente para ser utilizado por um único usuário final. Sendo assim, é inserido em ambientes únicos, Stand-alone, podendo ser implantando, por exemplo, em um tradicional computador de mesa (Desktop) ou um notebook. Em termos sistêmicos, podemos citar um sistema de vendas simples, destinado a uma loja de confecção, onde apenas uma única estação cliente se faz presente. O InterBase Desktop Edition só oferece suporte à plataforma Windows.

InterBase ToGo Edition

A edição ToGo pode ser entendida como a versão enxuta e portátil da Desktop Edition, sendo da mesma forma projetada para execução em ambientes Stand-alone. Até mesmo por sua característica portátil, a edição ToGo não contempla alguns dos recursos complementares da versão Desktop, tal como a ferramenta de administração IBConsole. Vendo pelo lado do desenvolvedor, a

ToGo é a versão embedded do InterBase. Numa tradução livre para o português, embedded seria algo como incorporado. Isso pode ser explicado da seguinte forma: em síntese, numa situação onde temos uma aplicação cliente acessando uma base de dados InterBase, haverão dois processos distintos em execução, sendo um relacionado à aplicação e outro ao banco. Na contramão, ao utilizar a versão ToGo, tudo é executado em um único processo, que é o da própria aplicação. Para que isso aconteça, são utilizadas DLL's da engine do InterBase ToGo que irão prover toda a funcionalidade do SGBD.

InterBase Developer Edition

Developer é a versão gratuita do InterBase. De forma simplificada, ela pode ser considerada uma versão limitada da edição Server. Diz-se limitada pelo fato dela prover todas as funcionalidades disponíveis na Server Edition, porém com algumas restrições. A principal é que a edição Developer destina-se única e exclusivamente para fins de desenvolvimento e estudo, não concedendo direitos para o seu uso em um ambiente de produção. Além disso, ela só poderá ser utilizada por aplicações clientes que rodam no mesmo computador, que neste caso também atuará como servidor. Dito isto, esta é a edição disponibilizada pela Embarcadero para que a comunidade de desenvolvedores possa elaborar seus protótipos antes de implantá-los e distribuí-los.

Ainda falando sobre as limitações, o InterBase Developer Edition oferece suporte até quatro núcleos de processamento, bem como 80 conexões simultâneas. Uma vez que o servidor é iniciado, 48 horas depois o mesmo não permite novas conexões, sendo necessário reiniciá-lo, caso haja realmente a necessidade por novas conexões. Outro ponto é o não suporte a add-ons disponíveis em outras versões, e que serão vistos a seguir.

Add-ons

Conforme mencionado, algumas das versões do InterBase provêm alguns Add-nos. Estes podem ser entendidos como pequenos recursos adicionais que são agregados ao produto. Estes recursos são providos através de alguns certificados de licença do próprio InterBase, sendo divididas basicamente em três categorias: criptografia, usuário e CPU. Na primeira, criptografia, temos a InterBase Strong Encryption Licence, que permite criptografia Strong (forte) AES para as edições Desktop, Server, e ToGo. Em outras palavras, se você precisa de uma criptografia neste nível para se utilizar com InterBase, você necessitará desta licença, do contrário, o padrão utilizado é uma criptografia Weak (fraca) DES (ver **Nota do DevMan 2**).

Na categoria de usuário, são apresentadas outras duas licenças. A primeira delas é a InterBase Simultaneous User Licence que permite a conexão simultânea de usuários numa estação que possua a edição Server Edition do produto. Nestes casos, deve-se adquirir um número de licenças igual ao número de usuários simultâneos que acessam o servidor. A outra licença relacionada é a InterBase Unlimited Users Licence, que abre margem para um número ilimitado de usuários que se conectam a um servidor InterBase (Server Edition). Um caso real onde fica explícita a necessidade por este quesito seria uma aplicação Web que se conecta diretamente ao servidor de banco de dados e que permite acesso irrestrito a utilizadores do mundo todo. Por fim, a licença relacionada ao processador (CPU) vem sob o nome de InterBase Additional CPUs Licence. Ela permite a adoção de CPU's adicionais até um limite máximo de 32, em contraste aos oito previstos em todos os outros certificados de licença.

Nota do DevMan 2

Data Encryption Standard, ou simplesmente DES, é um método de criptografia definido como uma norma padrão do governo norte-americano na década de 70. Até mesmo por este motivo, acabou sendo utilizada em grande escala a nível internacional. Tecnicamente este método trabalha com uma chave de 56-bits, sendo por este motivo considerado um método inseguro para muitos cenários de aplicações.

Advanced Encryption Standard (AES) é definido como um padrão de criptografia avançada e que também foi adotado pelo governo norte-americano. É considerado um sucessor natural do método DES, uma vez que seu algoritmo criptográfico substituiu seu predecessor, que havia sido violado. Seu trabalho se dá sobre um tamanho de bloco fixo de 128 bits, e uma chave que varia de 128 a 256 bits.

Embedded User Authentication (EUA)

Conforme já dito, o Firebird talvez seja o SGBD mais utilizado por desenvolvedores Delphi para a construção de suas aplicações Desktop. Apesar de atender grande parte dos requisitos, assim como qualquer outro produto ele não é totalmente perfeito. Uma das principais reclamações dos profissionais que utilizam o Firebird é sobre sua segurança. Isto porque, o SGBD armazena as credenciais de usuário e senha de seus bancos de dados em um arquivo separado, na verdade em um banco de dados paralelo, denominado security (ou security2, dependendo da versão). Dessa forma, independente das definições de usuário/senha que você estabeleça para o banco de dados de sua aplicação, qualquer pessoa que tiver acesso ao servidor e tomar posse do arquivo do banco, poderá acessá-lo. Para que isto aconteça, basta que se faça a cópia do mesmo para qualquer máquina que já possua o Firebird instalado e, portanto, com um arquivo security próprio. Isto resulta num acesso simples com as credenciais padrões de usuário e senha: SYSDBA e masterkey. Para solucionar este "problema", uma forma eficaz seria a do SGBD armazenar estas informações de segurança dentro de cada banco de dados, ao invés de armazenar todas as credenciais em um único lugar centralizado e separado. Ainda nos dias atuais este assunto ainda gera certa polêmica quando discutido, sempre havendo algum ponto de divergência entre os interessados e, portanto, não cabe entrar no mérito da questão.

A verdade é que, de forma inversa e inclusive fora do conhecimento do grande público, o InterBase já possui tal recurso desde suas versões mais anteriores, uma vez que este foi introduzido em sua versão 7.5. Ele vem sob o nome de Embedded User Authentication, ou simplesmente EUA, termo este que poderia ser traduzido livremente para algo como Autenticação de Usuário Embutida. Tentando elucidar o que foi dito, a **Figura 2** mostra a tela

de criação de uma nova base de dados da versão 7.5 do produto, lançada em 2004. No detalhe pode ser vista a opção de Embedded User Authentication já disponível na época.

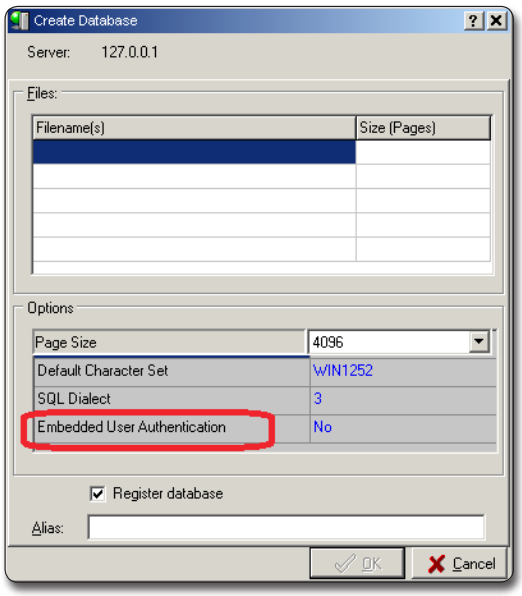


Figura 2. Opção de Embedded User Authentication no InterBase 7.5

Falando um pouco mais do EUA, o que este recurso faz é armazenar (embutir) o nome de usuário e senha do banco diretamente no próprio banco de dados, o que acaba por justificar sua nomenclatura. Consequentemente, esta característica proporciona alguns ganhos, tais como uma maior proteção da estrutura interna de metadados da base, bem como uma segurança maior no transporte do próprio banco, não podendo ser violada simplesmente pela substituição de um arquivo externo. Falando em termos práticos, a administração do EUA de uma base de dados se restringe única e exclusivamente ao seu proprietário (Owner), não dependente de um usuário SYSDBA. Todos os outros eventuais usuários do banco só podem alterar sua própria senha.

Além disso, o Embedded User Authentication está estritamente ligado a outro recurso disponível no InterBase XE, que é o de criptografia. Este recurso, conforme já mencionado, permite se estabelecer uma criptografia em nível de banco (suas páginas como um todo), ou mesmo de colunas específicas. Desta maneira, o recurso de criptografia só é possível para bancos que estejam com o EUA habilitado. Em outras palavras, o acesso a bancos InterBase criptografados só se torna possível para os usuários quando sua opção de EUA estiver definida como True. Da mesma forma que foi mostrada anteriormente, com o InterBase 7.5, na versão XE do produto esta opção fica disponível no momento da criação de um novo banco de dados (**Figura 3**).

A forma vista para se habilitar o EUA para uma base de dados, se deu através da utilização da ferramenta administrativa IBConsole, que acompanha a instalação do InterBase. Esta, sem dúvida, é a forma mais trivial de se habilitar o recurso. Contudo, opcionalmente, esta definição também pode ser feita através de linha de comando, por meio do utilitário iSQL (isql.exe, presente na basta Bin de instalação). Para um novo banco de dados, a habilitação do EUA é feito da seguinte forma:

```
CREATE DATABASE <nome do banco> WITH ADMIN OPTION
```

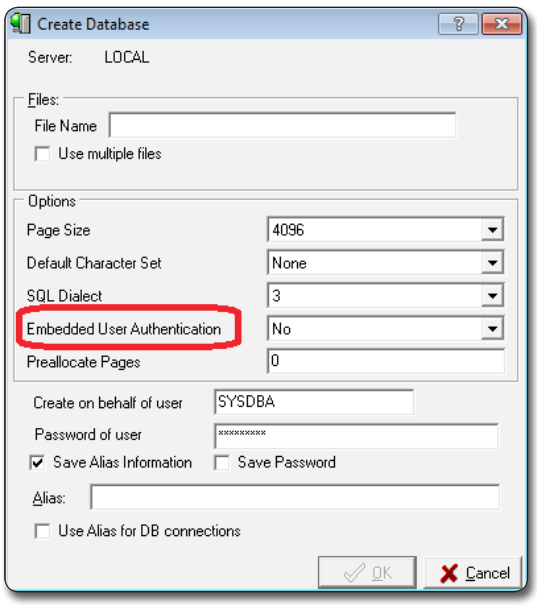


Figura 3. Opção de Embedded User Authentication no InterBase XE

Se comparado a um comando CREATE simples, a diferença fica por conta da indicação de WITH ADMIN OPTION. Internamente, esta cláusula determina que os dados de usuário e senha do banco sejam criadas em sua própria tabela de sistema RDB$USERSsystemTable. De forma similar, para os casos em que se deseja habilitar ou revogar o EUA para um banco de dados já existente, utiliza-se o comando ALTER, conforme o seguinte:

```
//habilita o recurso
ALTER DATABASE <nome do banco> ADD ADMIN OPTION

//desabilita o recuso
ALTER DATABASE <nome do banco> DROP ADMIN OPTION
```

Uma vez que o EUA é desabilitado, o acesso ao banco de dados volta ao processo padrão, onde a autenticação é realizada com base nos dados de usuário que ficam centralizados num banco de dados externo, aqui denominado Admin.ib, similar ao já citado security.fdb do Firebird.

Bancos de dados multiarquivos

O InterBase oferece suporte ainda a bancos de dados multi-arquivos (incluindo arquivos de sistema), recurso este também conhecido como Multifile Databases e que provê a adição de arquivos adicionais a um banco de dados. Citando a versão mais recente (XE) do produto, a definição de um banco multi-arquivos, partindo de um já existente, se dá através de um processo de Backup/Restore, via Database Restore da ferramenta IBConsole

ou via utilitário de comando gbak. Isso se justifica por esta ser a única forma de alterar a alocação de tamanho do arquivo de banco de dados, necessária ao processo.

A especificação do tamanho de cada arquivo adicional secundário ocorre de duas maneiras distintas. A primeira é indicando a página específica em que cada arquivo será iniciado. A segunda maneira é determinando o tamanho (Lenght) das páginas de banco de dados (database pages) de cada arquivo. Para este último caso, o próprio InterBase se encarrega de dinamizar o tamanho do arquivo final envolvido. Em termos de código, a sintaxe de definição para cada uma das opções é vista a seguir, tendo em vista a utilização do utilitário iSQL.

```
CONNECT 'banco_de_dados_principal'
ALTER DATABASE
ADD FILE 'arquivo_de_banco_de_dados_secundario' STARTING AT valor_pagina;

CONNECT 'banco_de_dados_principal'
ALTER DATABASE
ADD FILE 'arquivo_de_banco_de_dados_secundario' LENGTH valor_pagina;
```

Preallocate Pages

Citando novamente o Interbase XE, no momento da criação de um novo banco de dados através de sua ferramenta administrativa IBConsole, é possível notar a presença de outra opção adicional, além da Embedded User Authentication. Conforme pode ser visto em destaque na **Figura 4**, a opção em questão é a denominada Preallocate Pages.

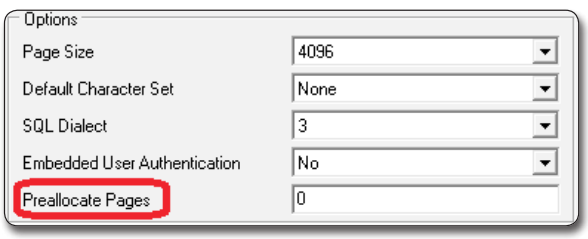


Figura 4. Opção Preallocate Pages

Explicando melhor, Preallocate Pages é o recurso do InterBase que permite especificar um espaço adicional extra no novo banco de dados que está sendo criado. Este processo de pré-alocação de páginas do banco se correlaciona diretamente com o recurso de Multifile Databases, uma vez que oferece suporte aos arquivos secundários de banco de dados. Dessa forma, o espaço pré-alocado é distribuído a todos estes arquivos adicionais, com base nas especificações de tamanho de cada um. Por padrão, o recurso de Preallocate Pages não vem especificado num banco de dados InterBase que está sendo criado, ou seja, nenhuma página de banco é pré-alocada nativamente. Do contrário, isto poderia causar algum transtorno ou desperdício, em situações em que não haja a necessidade de reserva de espaço na base.

Devido a uma falha humana, por exemplo, caso a definição do recurso de Preallocate exceda o próprio espaço disponível em disco, o InterBase aguarda a liberação do espaço necessário para concluir a pré-alocação. Caso isto não aconteça, o espaço extra não é alocado. Isto impede que se configure algo que o próprio hardware envolvido não irá suportar.

Conclusão

Desde muito tempo é quase que natural estabelecer o Firebird como sendo o SGBD mais comumem termos de utilização quando se trata de aplicações Desktop desenvolvidas em Delphi. Este cenário tende a perdurar ainda por um longo tempo, contudo, em meio a esse estigma, estão inseridos outros SGBD's, tais como Microsoft SQL Server, Oracle e o InterBase. Sendo assim, sobre este último, o presente artigo foi proposto com a intenção de inseri-lo novamente no contexto Delphi, mesmo que somente de forma conceitual. De certa forma, mostrar ao desenvolvedor que o InterBase continua existindo , com a robustez e eficácia de seus outros concorrentes, contando inclusive com uma versão free. Outro fator a se destacar é que, uma vez que se surja a necessidade em se trabalhar com o produto, o Delphi conta com uma gama de componentes e driver nativos exclusivos, o que pode tornar o desenvolvimento ainda mais facilitado.

Por se tratar de um assunto que há muito tempo não era mais abordado nesta revista, sinta-se livre em tirar suas próprias conclusões, fazer seus testes a aumentar seu conhecimento. Na seção **Links**, é disponibilizado o endereço oficial do produto onde você irá encontrar, entre outras coisas, a documentação oficial do InterBase XE, bem como sua seção de downloads, onde é possível baixar sua versão gratuita (Developer Edition). Novamente reforçando, na próxima parte do artigo, veremos mais conceitos que envolvem o InterBase, bem como a introdução de uma parte mais prática relacionada ao assunto.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 6 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas.
Blog: prismsilver.blogspot.com

Embarcadero InterBase XE – Página oficial do produto
http://embarcadero.com/products/interbase

Embarcadero InterBase XE – Download
https://downloads.embarcadero.com/free/interbase

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Trabalhando com Streams e Threads

Crie um gerenciador de downloads

A informação é um fator extremamente importante para todos em qualquer contexto, seja na informática, nos relacionamentos, estudo, trabalho, etc. No ambiente computacional e, mais especificamente, dentro do desenvolvimento de Software existem vários meios para a manipulação das informações, que podem ser simples através de uma rápida e objetiva função, ou mais complexas, onde demandam maiores recursos. Outro fator importante é com relação à troca das informações entre as entidades envolvidas, como é o caso das redes e em especial, a Internet, onde mecanismos devem ser implementados para que todo o processo ocorra de forma satisfatória. A comunicação no ambiente computacional tem como um de seus pilares os protocolos, onde o TCP (Transfer Control Protocol) merece ser evidenciado, uma vez que fornece a garantia de que as informações serão devidamente entregues ao seu destinatário, porém, além deste, uma variedade de tecnologias podem ser empregadas quando surge a necessidade de executar uma tarefa mais específica, como é o caso da transferência de um arquivo binário por exemplo. Um recurso que pode ser adotado para a manipulação e transferência de informações de uma forma mais ampla é o Stream. Este pode ser resumido como um fluxo de dados presente na memória, que tem um início e um fim, sendo constituído por uma sequência de bytes, permitindo que a aplicação possa percorrer tais dados obtendo informações como tamanho, por exemplo, tendo um funcionamento semelhante ao processo de navegação pelos registros de um DataSet. Além disso, os Streams podem ser utilizados para manipular uma diversidade de tipos de dados, como imagens, áudio, binários, textos e assim por diante, bem como ter seu tamanho estendido até os limites disponíveis e permitidos pela memória física.

No cenário de Streams é muito comum ouvir o termo Streaming. Como é uma palavra inglesa no gerúndio, indica que é uma ação que está ocorrendo,

Resumo DevMan

De que se trata o artigo:

Este artigo aborda a utilização de Streams, sequências de bytes criados na memória que podem ser empregados para ler, gravar e copiar informações, oferecendo recursos adicionais para a manipulação dos dados, apresentando também uma forma de empregá-los na elaboração de uma simples ferramenta de gerenciamento de Downloads em segmentos, baseada em Threads, que permitem a execução de tarefas em paralelo.

Em que situação o tema é útil:

Manipular informações em nível de bytes, podendo copiar arquivos ou dados de uma forma direta ou mais elaborada, podendo ainda "recortar" tais dados e empregar o conceito de concorrência nos projetos através da utilização de Threads.

Streams:

A informação é um dos pilares no ambiente computacional. Todos os dias, mais e mais dados são lidos, copiados e compartilhados entre os usuários. Ter à disposição um mecanismo capaz de oferecer diversos recursos para a manipulação das mais variadas informações, torna-se algo essencial em grande parte das aplicações, e uma tecnologia que pode ser empregada para este fim é o Stream. Ele pode ser resumido como uma sequência de bytes presente na memória, que oferece ao desenvolvedor a possibilidade de copiar, ler e gravar as informações, enviando-as para um destino desejado, que pode ser outro Stream ou um meio físico de armazenamento, como o HD. Sendo assim, a aplicação pode "quebrar" arquivos binários e posteriormente, reuni-los novamente, como é o caso de um Software gerenciador de Downloads. O Delphi conta com uma variedade de classes que podem ser empregadas para as mais diversas tarefas empregando Streams, que por sua vez possuem muitos métodos importantes. Neste artigo, serão apresentadas as classes básicas para a utilização deste recurso, empregando técnicas para dividir um arquivo qualquer em três partes sem corrompê-lo e assim efetuar o Download do mesmo paralelamente, recorrendo também ao uso de Threads.

ou em outras palavras, o momento em que um determinado Stream está sendo utilizado para a transferência das informações, podendo ser definido então como um fluxo constante de dados. Este processo é muito empregado na Internet principalmente em áudio e vídeo, uma vez que tal recurso permite que determinado arquivo possa ter sua reprodução iniciada sem que o mesmo tenha que ser obtido completamente, o que poderia levar um tempo considerável se o arquivo em questão for grande. Contudo, dentro deste contexto de Streaming para redes como a Internet, existem outros conceitos e técnicas que são empregados, tais como protocolos específicos, otimização de transmissão, entre outros.

Outro recurso que é comum de ser adotado quando Streams são empregados é o Buffer. Ele por sua vez, também está contido em uma área da memória, podendo ser utilizado para a leitura e gravação dos dados, porém ele tem a função de armazenar determinada informação temporariamente, informação esta que foi produzida, mas que ainda não foi consumida. Sendo assim, a utilização de Buffers também é muito útil quando a velocidade da transferência de dados entre as entidades envolvidas é diferente ou podem ocorrer momentos de pausa na troca, como é o caso de aplicações de multimídia rodando sobre a Internet.

Com relação ao Delphi, desde suas versões anteriores, a classe base empregada na utilização de Streams é a TStream, permitindo que os dados possam ser lidos, copiados ou gravados para diversos meios, tais como memória dinâmica, memória estática, etc. Também vale a pena citar que esta classe é do tipo abstrata, que por sua vez, é constituída por alguns métodos que não possuem uma definição (virtual) e devem ser elaborados nas classes derivadas, ou em outras palavras, a classe abstrata é a definição de determinado objeto de forma incompleta. Sendo assim, existem outras classes que descendem de TStream, onde cada uma é direcionada a manipulação de determinado tipo de dado e implementam funções mais especificas para seu objetivo. A **Tabela 1** apresenta essas classes derivadas.

As classes derivadas de TStream possuem diversos métodos para que o desenvolvedor possa empregá-los em diferentes momentos, podendo manipular os dados, alguns comuns entre todas as subclasses e outros mais específicos. Outro ponto importante relacionado à manipulação de Streams é que ao adotar este recurso, o desenvolvedor pode através de sua aplicação, dividir a informação em partes, gravando-as de forma separada, ou ainda unir uma informação a outra. A **Tabela 2** apresenta alguns dos principais métodos empregados na criação e manipulação de Streams.

Além dos métodos, as classes descendentes de TStream possuem algumas propriedades que são importantes, tais como Size, que armazena o tamanho atual de um Stream, e Position, que indica a posição atual.

Contudo, em um bloco de código podem ser empregadas instâncias de classes diferentes derivadas de TStream que podem compartilhar informações entre si, em outras palavras, um objeto derivado de TFileStream por exemplo, pode ler os dados presentes em um outro objeto que foi derivado da classe TStringStream, e assim gravá-los efetivamente em um meio físico.

Classe	Descrição
TFileStream	Destinada à manipulação de arquivos, permitindo que as aplicações possam ler e gravar os mesmos em um meio físico.
TMemoryStream	Empregada para a manipulação de dados na memória dinâmica, capacitada também a oferecer recursos de I/O (Entrada e Saída).
TStringStream	Destinada à manipulação de Strings, oferecendo recursos de I/O e permitindo a gravação das informações diretamente para um meio físico.
TOleStream	Utilizada para interagir com uma interface COM para escrita e leitura de um objeto OLE.
TBlobStream	Empregada em manipulações com campos do tipo BLOB, ou em outras palavras, grandes objetos do tipo binário.
TWinSocketStream	Utilizada para escrever e ler informações através de uma conexão de soquete (Socket).
THandleStream	Destina-se à leitura e gravação de soquetes, arquivos, entre outros, empregando um identificador (Handle).

Tabela 1. Classes descendentes de TStream

Método	Descrição
Create	O método construtor deve ser utilizado para a criação de uma instância da classe em questão. Na classe TFileStream, alguns parâmetros podem ser especificados, tais como o fmCreate, indicando que o arquivo deve ser criado ou substituído.
SetSize	Permite a definição do tamanho (Bytes) de um Stream.
Read	Utilizado para a leitura dos dados de um Stream para um Buffer, retornando o número de Bytes lidos.
Write	Utilizado para a escrita de dados presentes em um buffer para um Stream, retornando o número de Bytes gravados.
Seek	Define a posição atual para um Stream, permitindo assim se mover pelo mesmo.
CopyFrom	Empregado para copiar os dados de um Stream para outro de uma forma simples.
ReadBuffer	Efetua a leitura dos dados de Stream para um Buffer.
WriteBuffer	Efetua a gravação dos dados de Stream para um Buffer.

Tabela 2. Métodos importantes para a manipulação de Streams no Delphi

Tutorial

Trabalhando com Streams na prática

Para dar início a uma aplicação utilizando Streams, será criado um novo projeto (File>New>VCL Forms Application) com o intuito de desenvolver um simples Software capaz de gerenciar Downloads, obtendo os arquivos de forma paralela, dividindo-os em três partes e posteriormente, unindo-os em um único arquivo. No formulário principal da aplicação, que foi denominado FrPrincipal, foram adicionados dois Edits, cinco Labels, um GroupBox, um SpeedButton e três ProgressBar. Neste exemplo, também será empregado o recurso de Threads, uma vez que os Downloads serão efetuados concorrentemente, além do fato da janela principal da aplicação permanecer disponível enquanto as tarefas são realizadas. Uma Thread pode ser definida como a unidade básica envolvida na execução de uma aplicação. Sendo assim, quando um projeto é executado no Delphi, uma Thread também é iniciada, contendo o processo principal do software em questão. Dentro deste contexto, existem as aplicações MonoThreads, onde os códigos são executados de forma sequencial, e as MultiThreads, onde existem várias Threads que podem executar códigos distintos paralelamente. No Delphi, a classe básica para implementação de Threads é a TThread, que por sua vez, pode ser facilmente herdada. Ao utilizar esta classe o desenvolvedor também deve estender ou sobrescrever o método abstrato Execute, definindo assim o código que a mesma deverá executar no momento em que for instanciada e inicializada. Entretanto, a classe TThread possui outros métodos assim como propriedades que também podem ser utilizados para diversas finalidades. A **Tabela 3** apresenta os principais métodos e propriedades da classe TThread.

Como pode ser observado, a classe TThread conta com diversos métodos e propriedades, onde tais métodos podem ser facilmente estendidos.

Para dar continuidade ao projeto, será criada uma classe derivada de TThread com o nome de TDownloadThread, que por sua vez, terá todos os métodos e propriedades que serão utilizados para o Download segmentado, devendo esta ser definida logo após a concepção do formulário principal. Para efetuar o processo de Download, o componente TIdHTTP, presente na Unit IdHTTP e pertencente a paleta Indy Clients, será criado em tempo de execução. A classe TIdHTTP, descendente de TIdCustomHTTP, permite a implementação de um cliente para o protocolo HTTP (HyperText Transfer Protocol), oferecendo a base necessária para a transmissão das informações através de uma rede, como é o caso da Internet. Esta classe também conta com os métodos WorkBegin, Work e WorkEnd, utilizados para a execução de blocos de código no início, durante e fim do processo de Download, sendo os mesmos também implementados através da classe TDownloadThread. A **Listagem 1** apresenta o código da classe TDownloadThread.

No código da **Listagem 1** temos uma sequência de variáveis utilizadas para a criação da Thread principal. Seguindo o contexto de criação, a propriedade FInicio tem o objetivo de guardar a posição inicial para o Download do segmento, FTamanhoTotal deve armazenar o tamanho total do arquivo que será feito o Download, FLimiteDownload guardará o tamanho para o segmento, FOrdemParte indica a ordem dos segmentos (primeiro, segundo e etc.), FTotalPartes será responsável pelo total de segmentos para o Download (neste caso serão três), FArquivoDownload é responsável pela URL (Uniform Resource Locator) para Download do arquivo, FIndyHTTP é um objeto do tipo TIdHTTP responsável por efetuar a conexão, FStreamMemoria é um objeto do tipo Stream e FFormulário tem o objetivo de armazenar qual o formulário principal, que futuramente, será efetuada a sincronização com a Thread. Também foram definidos os métodos WorkBegin, para gerenciar o que deve ser feito no momento em que o download

Método/Propriedade	Descrição
Create	O método construtor, responsável pela criação de uma instância da classe que pode ser estendido para se adaptar as necessidades do projeto. Neste momento, também se define se a Thread deve ser imediatamente inicializada, através do parâmetro de entrada CreateSuspended do tipo Boolean.
Resume	Se no momento da criação da Thread o parâmetro CreateSuspended foi definido como verdadeiro (Create(True)), então este método pode ser utilizado para que a Thread inicie sua execução no instante em que é chamado.
Execute	É o método que deve conter o código que será executado pela Thread, onde a propriedade Terminated pode ser verificada para constatar se a execução da Thread foi concluída.
Suspend	Uma vez que a Thread esteja em execução, este método permite que a mesma seja pausada, podendo sua execução ser iniciada posteriormente.
Synchronize	Este método tem o objetivo de sincronizar a Thread em questão com a Thread principal da aplicação, efetuando uma chamada a determinado procedimento externo. Convém empregá-lo toda vez que é efetuada uma chamada a um método que não faz parte da Thread.
OnTerminate	Este método é chamado no momento em que a Thread é finalizada.
Terminate	Este método encerra a execução da Thread em questão.
Priority	Esta propriedade permite a definição de um nível de prioridade para a execução da Thread em relação a outras Threads que estão sendo executadas, onde quanto maior o nível, maior o tempo de processamento dedicado a mesma.
FreeOnTerminate	Permite que o objeto instanciado da classe TThread em questão seja liberado da memória assim que for encerrado, caso seu valor seja definido como verdadeiro.

Tabela 3. Propriedades e Métodos importantes da classe TThread do Delphi

iniciar, o Work, responsável pelo código que será executado durante o processo, e o WorkEnd, para o momento da conclusão do Download. Vale lembrar que estes métodos nada mais são do que eventos do próprio componente. Para finalizar, será sobrescrito o método Execute (override), alterado o método Create e criada uma função, GetStreamMemoria, com o intuito de retornar o conteúdo do objeto Stream.

O formulário principal do projeto também recebeu dois novos métodos, declarados abaixo da seção public, que são apresentados a seguir:

```
procedure ConclusaoProcesso;
procedure CopiarStream(const StreamOrigem, StreamDestino: TStream);
```

O primeiro é o método que será chamado pelas instâncias da classe TDownloadThread com o intuito de efetuar a conclusão do processo de Download, e o segundo, será utilizado para efetuar a cópia de um Stream para outro.

O formulário principal recebeu a inclusão de novas variáveis para o processo, sendo elas:

```
TDownloadParte1: TDownloadThread;
TDownloadParte2: TDownloadThread;
TDownloadParte3: TDownloadThread;
ThreadsConcluidas: Integer;
URLArquivo, EnderecoGravar: string;
```

Concluída a definição das classes, métodos, propriedades e variáveis, a próxima etapa é iniciar a implementação dos códigos. O primeiro método definido foi o construtor Create para a classe TDownloadThread. A **Listagem 2** apresenta o código do método Create.

É possível notar que o construtor é utilizado para a definição de algumas propriedades internas para a instancia de TDownloadThread em questão (FArquivoDownload, FFormulario e etc), bem como o mesmo tem a propriedade CreateSuspended definida como False, o que ocasiona a execução da Thread a partir deste momento.

O próximo método implementado é o WorkBegin, que por sua vez é executado toda vez que o processo de Download for iniciado. A **Listagem 3** apresenta sua implementação.

Este método é apenas utilizado para definir o valor inicial (0) para os valores das barras de progresso (TProgressBar), presentes no formulário principal (FFormulario), através da propriedade Position, de acordo com uma das três Threads que estão em execução, utilizando a propriedade FOrdemParte de TDownload Thread que armazena qual é a parte referente a Thread, uma vez que foi adotado o download segmentado em três partes.

A seguir é definido o código para o método Work, responsável por executar determinado trecho de código durante o processo de download. A **Listagem 4** apresenta o código para o método Work.

Este método é utilizado para atualizar os objetos instanciados de TProgressBar de acordo com o progresso do Download através da propriedade AWorkCount, que armazena o progresso.

Listagem 1. Classe TDownloadThread

```
TDownloadThread = class(TThread)
private
  FInicio: Integer;
  FTamanhoTotal: Integer;
  FLimiteDownload: Integer;
  FOrdemParte: Integer;
  FTotalPartes: Integer;
  FArquivoDownload: string;
  FIndyHTTP: TIdHTTP;
  FStreamMemoria: TStream;
  FFormulario: TFrPrincipal;
  procedure WorkBegin(ASender: TObject; AWorkMode: TWorkMode;
  AWorkCountMax: Int64);
  procedure Work(ASender: TObject; AWorkMode: TWorkMode;
  AWorkCount: Int64);
  procedure WorkEnd(ASender: TObject; AWorkMode: TWorkMode);
protected
  procedure Execute; override;
public
  constructor Create(ArquivoDownload: string; Formulario: TFrPrincipal;
   OrdemParte, TotalPartes: Integer);
  function GetStreamMemoria: TStream;
end;
```

Listagem 2. Método construtor Create da classe TDownloadThread

```
constructor TDownloadThread.Create(ArquivoDownload: string; Formulario:
TFrPrincipal; OrdemParte, TotalPartes: Integer);
begin
  FArquivoDownload := ArquivoDownload;
  FFormulario := Formulario;
  FOrdemParte := OrdemParte;
  FTotalPartes := TotalPartes;
  inherited Create(False);
end;
```

Listagem 3. Método WorkBegin da classe TDownloadThread

```
procedure TDownloadThread.WorkBegin(ASender: TObject;
AWorkMode: TWorkMode;
 AWorkCountMax: Int64);
begin
  if FOrdemParte = 1 then
    FFormulario.ProgressBar1.Position := 0
  else if FOrdemParte = 2 then
    FFormulario.ProgressBar2.Position := 0
  else if FOrdemParte = 3 then
    FFormulario.ProgressBar3.Position := 0;
end;
```

Na linha 6, FLimiteDownload, possui o tamanho total para o segmento. Também é utilizada a função Trunc com o objetivo de truncar um valor real para inteiro, atribuindo o mesmo a propriedade Position da TProgressBar, permitindo assim o cálculo das informações em porcentagem. Os objetos TLabels referentes a cada segmentação de download também possuem sua propriedade Caption atualizada para exibir informações sobre o progresso (linhas 12, 17 e 22), onde o primeiro valor é referente ao byte inicial do segmento em questão, o segundo, referente ao byte final, e o terceiro, a porcentagem que foi concluída até o momento.

Após a definição dos métodos WorkBegin e Work, resta também a definição do WorkEnd, que por sua vez é executado no encerramento do Download através de um objeto do tipo TIdHTTP. A **Listagem 5** apresenta o código empregado para o método WorkEnd da classe TDownloadThread.

Listagem 4. Método Work da classe TDownloadThread

```
01 procedure TDownloadThread.Work(ASender: TObject;
     AWorkMode: TWorkMode;  AWorkCount: Int64);
02 var
03   Porcentagem: Integer;
04   Descricao: string;
05 begin
06   Porcentagem := Trunc((AWorkCount/FLimiteDownload)*100);
07   Descricao := IntToStr(FInicio) + '-' + IntToStr(FInicio + FLimiteDownload) + ' / ' +
08             IntToStr(Porcentagem) + '%';
09   if FOrdemParte = 1 then
10   begin
11     FFormulario.ProgressBar1.Position := Porcentagem;
12     FFormulario.Label1.Caption := Descricao;
13   end
14   else if FOrdemParte = 2 then
15   begin
16     FFormulario.ProgressBar2.Position := Porcentagem;
17     FFormulario.Label2.Caption := Descricao;
18   end
19   else if FOrdemParte = 3 then
20   begin
21     FFormulario.ProgressBar3.Position := Porcentagem;
22     FFormulario.Label3.Caption := Descricao;
23   end;
24 end;
```

Listagem 5. Método WorkEnd da classe TDownloadThread

```
procedure TDownloadThread.WorkEnd(ASender: TObject;
AWorkMode: TWorkMode);
begin
  if FOrdemParte = 1 then
    FFormulario.ProgressBar1.Position := 100
  else if FOrdemParte = 2 then
    FFormulario.ProgressBar2.Position := 100
  else if FOrdemParte = 3 then
    FFormulario.ProgressBar3.Position := 100;
end;
```

Este simples método é utilizado para garantir e definir o valor final (100) para as barras de progresso também de acordo com a propriedade FOrdemParte da Thread em questão.

Com a criação de alguns métodos auxiliares, a próxima etapa é definir o método principal de TDownloadThread, o método Execute. A **Listagem 6** apresenta seu código.

O primeiro passo deste método é a criação de uma instância da classe TIdHTTP (linha 3), que por sua vez foi atribuída ao objeto FIndyHTTP. Em seguida, são definidas algumas propriedades para o mesmo, para o correto funcionamento do processo de Download (linhas 5-12). Primeiramente, são atribuídos os eventos, OnWorkBegin, OnWork e OnWorkEnd para os devidos métodos criados anteriormente, WorkBegin, Work e WorkEnd. Posteriormente, o método Head do objeto FIndyHTTP é utilizado para obter informações do arquivo que será efetuado o Download, tal como o tamanho total do mesmo, entre outras.

Listagem 6. Método Execute classe TDownloadThread

```
01 procedure TDownloadThread.Execute;
02 begin
03   FIndyHTTP := TIdHTTP.Create(FFormulario);
04   try
05     with FIndyHTTP do
06     begin
07       OnWorkBegin := WorkBegin;
08       OnWork := Work;
09       OnWorkEnd := WorkEnd;
10       Head(FArquivoDownload);
11       HandleRedirects := True;
12     end;
13     FTamanhoTotal := FIndyHTTP.Response.ContentLength;
14     if not Pos('bytes', FIndyHTTP.Response.AcceptRanges) > 0 then
15     raise Exception.Create('Segmentação não aceita para este download!');
16     FLimiteDownload := FTamanhoTotal div FTotalPartes;
17     FInicio := FLimiteDownload * (FOrdemParte - 1);
18     if FOrdemParte = FTotalPartes then
19       Inc(FLimiteDownload, FTamanhoTotal mod FTotalPartes);
20     FStreamMemoria := TMemoryStream.Create;
21     FStreamMemoria.Position := FInicio;
22     with FIndyHTTP do
23     begin
24       Request.ContentRangeStart := FInicio;
25       Request.ContentRangeEnd := FInicio + FLimiteDownload;
26       Request.Range := Format('%d-%d',[FInicio,FInicio + FLimiteDownload]);
27     end;
28     FIndyHTTP.Get(FArquivoDownload, FStreamMemoria);
29     FStreamMemoria.Size := FInicio + FLimiteDownload;
30     FStreamMemoria.Position := FInicio;
31     FreeOnTerminate := True;
32     Synchronize(FFormulario.ConclusaoProcesso);
33   finally
34     FIndyHTTP.Disconnect;
35     FIndyHTTP.Free;
36   end;
37 end;
```

A propriedade HandleRedirects é definida como verdadeiro para a manipulação automática de redirecionamentos de URL pelo objeto FIndyHTTP, caso ocorram. Em seguida na linha 13, a propriedade FTamanhoTotal recebe a informação presente na propriedade Response.ContentLength, que por sua vez foi adquirida com a utilização do método Head e assim, armazena o tamanho total do arquivo que será efetuado o Download. Um ponto importante neste momento é saber se a segmentação é permitida para a origem do Download, onde a não utilização deste teste poderia ocasionar uma falha durante o processo de segmentação. Este teste pode ser efetuado através da função Pos (linha 14), verificando se consta no retorno presente em Response.AcceptRanges, os caracteres "Bytes", indicando que podem ser especificadas faixas (segmentos) para o processo de Download, e caso não sejam encontrados tais caracteres, uma exceção é gerada com uma mensagem para abortar a execução. Uma vez que a segmentação pode ser empregada, a propriedade FLimiteDownload recebe o resultado da divisão do tamanho total pelo total de partes (FTamanhoTotal div

FTotalPartes) (linha 16), com o objetivo de definir qual será o tamanho que cada parte deve conter. Em seguida, a propriedade FInicio obtém o resultado da multiplicação da propriedade FLimiteDownload pela expressão (FOrdemParte – 1), pois para o primeiro segmento o valor inicial deve ser 0, o segundo, o valor deve ser o final do primeiro (levando em consideração o valor de FLimiteDownload), e assim por diante. Continuando, se a propriedade FOrdemParte for equivalente a FTotalPartes, indicando que é a última parte, é efetuada uma inclusão (ou não) em FLimiteDownload para o devido ajuste da última parte, uma vez que pode restar 1 byte de diferença, por exemplo, entre ela e as outras partes. Em seguida na linha 20, uma instância da classe TMemoryStream é criada e atribuída ao objeto FStreamMemoria. A classe TMemoryStream é direcionada à manipulação de diversos tipos de dados diretamente na memória dinâmica, sendo ideal para ser adotada nesta etapa temporária do processo para a recepção dos dados. Também é definida a posição atual para o objeto FStreamMemoria através da propriedade Position. Outro ponto extremamente importante e essencial para o processo de segmentação do Download é a definição das subpropriedades ContentRangeStart, Content RangeEnd e Range, presentes na propriedade Request (linhas 22-27). Uma vez que o Download será segmentado, para cada parte é necessário definir onde a mesma se inicia e finaliza. Em ContentRangeStart é definido o valor inicial para o segmento, em ContentRangeEnd o valor final, e em Range, o tamanho total da parte, que neste exemplo é definido com base no início (FInicio) e no tamanho (FInicio + FLimiteDownload). Também é utilizada a função Format para a devida formatação dos valores para a subpropriedade Range. Definidas todas as propriedades necessárias, o Download é iniciado na Thread através do método Get do objeto FIndyHTTP (linha 28), que por sua vez recebe como parâmetros, a URL do arquivo de Download e o Stream responsável por receber os dados. A seguir, linha 29, é definido o tamanho para o objeto FStreamMemoria através da propriedade Size, bem como a posição atual para a manipulação de dados do mesmo através de Position (linha 30). Para concluir, é feita a sincronização (linha 32) com o método ConclusaoProcesso, presente no formulário principal (FFormulario), fazendo assim com que seja realizada a apuração do resultado, sendo também definida a propriedade FreeOnTerminate da Thread para verdadeiro para que ocorra sua liberação da memória, liberando assim os outros objetos envolvidos através do método Free.

Antes de criar o método de conclusão do Download, será definida a função auxiliar declarada anteriormente, GetStreamMemoria, para a classe TDownloadThread, com o intuito de acessar o objeto interno FStreamMemoria. A **Listagem 7** demonstra o código para a função GetStreamMemoria.

Esta é uma simples função que retorna o valor do objeto FStreamMemoria.

Com a função auxiliar criada, será definido o outro método auxiliar, CopiarStream, para o formulário principal. A **Listagem 8** apresenta o código do método CopiarStream.

Listagem 7. Função GetStreamMemoria da classe TDownloadThread

```
function TDownloadThread.GetStreamMemoria: TStream;
begin
  Result := FStreamMemoria;
end;
```

Listagem 8. Método CopiarStream

```
procedure TFrPrincipal.CopiarStream(const StreamOrigem,
StreamDestino: TStream);
var
  Buffer: array[0..1023] of Byte;
  BytesLidos: Integer;
begin
  repeat
    BytesLidos := StreamOrigem.Read(Buffer, 1024);
    StreamDestino.Write(Buffer, BytesLidos);
  until BytesLidos = 0;
  StreamOrigem.Free;
end;
```

De início, este método recebe como parâmetros, o Stream de origem e o Stream de destino (TStream). Na seção de declaração de variáveis, é definido um Array do tipo Byte de tamanho 1024, que recebeu a denominação de Buffer e em seguida, outra definida como BytesLidos do tipo Integer para auxiliar durante o processo. A estrutura repeat é empregada para executar o trecho de código até que não existam bytes a serem lidos e gravados, onde o processo de leitura é efetuado através da função Read do objeto StreamOrigem e com a utilização do Buffer, assim como o processo de gravação é realizado pelo objeto de destino (StreamDestino) através do método Write em conjunto com o Buffer e os BytesLidos. Para concluir, o StreamOrigem é liberado da memória.

A próxima etapa é criar o método final para o processo que vai efetuar a cópia dos dados presentes nos objetos instanciados de TMemoryStream (FStreamMemoria) para um objeto instanciado da classe TFileStream, fazendo assim a união dos segmentos e a gravação dos dados para um arquivo em disco. A **Listagem 9** apresenta o código do método ConclusaoProcesso.

Nesta etapa, quando uma Thread é encerrada, o contador (ThreadsConcluidas) é incrementado através de Inc. Uma vez que todas as Threads estejam finalizadas, através da verificação da variável ThreadsConcluidas, que deve ser equivalente a três, uma instância da classe TFileStream é criada e atribuída ao objeto ArquivoStream, contendo o local onde o arquivo será gravado (EnderecoGravar). Ela também recebe o parâmetro fmCreate, indicando que o arquivo deve ser criado ou sobrescrito. Posteriormente, o método CopiarStream é chamado para copiar o conteúdo de cada objeto TMemoryStream presente em cada Thread instanciada de TDownloadThread, com o auxílio da função GetStreamMemoria diretamente para o objeto ArquivoStream (TFileStream), e evidentemente, o arquivo no meio físico informado. Terminado este processo, o objeto ArquivoStream é liberado da memória. Para concluir, é efetuada uma pergunta ao usuário se o mesmo deseja abrir o arquivo que foi obtido e, caso o retorno da mensagem seja verdadeiro,

então a função ShellExecute, presente na Unit Winapi.ShellAPI, é chamada para executar o mesmo, recebendo com parâmetros o Handle do formulário, a chamada open (abrir), a conversão do endereço que está em String para PChar e um último parâmetro, SW_SHOWNORMAL, indicando que o mesmo deve ser exibido sem alterações de exibição.

Para finalizar o processo, basta inserir o código que iniciará o processo de Download no SpeedButton1 adicionado anteriormente ao formulário principal. A **Listagem 10** apresenta o código do evento OnClick do SpeedButton1 para iniciar o processo de Download segmentado.

Neste simples trecho de código, a variável declarada anteriormente no formulário principal, URLArquivo, recebe o valor da propriedade Text do objeto edUrl, assim como a variável EnderecoGravar recebe o valor da propriedade Text do edSalvarEm. Em seguida, cada TDownloadParte derivada de TDownloadThread

Listagem 9. Método ConclusaoProcesso

```
procedure TFrPrincipal.ConclusaoProcesso;
var
  ArquivoStream: TFileStream;
begin
  Inc(ThreadsConcluidas);
  if (ThreadsConcluidas = 3) then
  begin
    ArquivoStream := TFileStream.Create(EnderecoGravar, fmCreate);
    try
      CopiarStream(TDownloadParte1.GetStreamMemoria, ArquivoStream);
      CopiarStream(TDownloadParte2.GetStreamMemoria, ArquivoStream);
      CopiarStream(TDownloadParte3.GetStreamMemoria, ArquivoStream);
    finally
      ArquivoStream.Free;
      if Application.MessageBox('Processo Concluído!
       Deseja abrir o arquivo?','Download', 32 + 4) = mrYes then
        ShellExecute(handle,'open',PChar(EnderecoGravar), '','',SW_SHOWNORMAL);
    end;
  end;
end;
```

Listagem 10. Código do evento OnClick() do SpeedButton1

```
procedure TFrPrincipal.btIniciarDownloadClick(Sender: TObject);
begin
  URLArquivo := edUrl.Text;
  EnderecoGravar := edSalvarEm.Text;
  TDownloadParte1 := TDownloadThread.Create(URLArquivo, Self, 1, 3);
  TDownloadParte2 := TDownloadThread.Create(URLArquivo, Self, 2, 3);
  TDownloadParte3 := TDownloadThread.Create(URLArquivo, Self, 3, 3);
end;
```

é devidamente instanciada, recebendo como parâmetros o endereço para Download (URL), o formulário para a sincronização (FrPrincipal), a parte (ou ordem) referente a Thread em questão e o total de partes.

Neste momento, a aplicação de exemplo para Download de arquivos de forma segmentada está concluída. A **Figura 1** apresenta o visual final da aplicação efetuando o Download de um executável através da Internet.

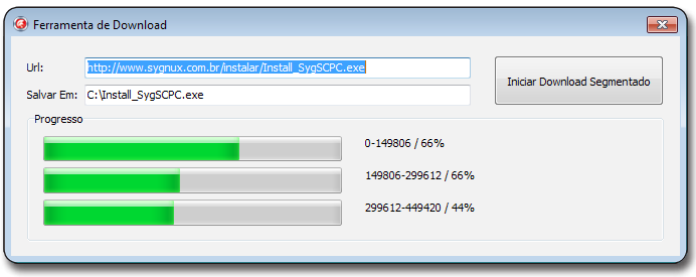


Figura 1. Visual final da aplicação de gerenciamento de download

Conclusão

O emprego de Streams para as mais diversas finalidades é um processo que não é tão complexo através do Delphi e que por sua vez oferece diversas classes com várias finalidades e funcionalidades para a manipulação deste recurso. Entretanto, por ter a possibilidade de lidar diretamente com Bytes, deve-se sempre agir com cautela para não ocasionar uma corrupção nos dados, uma vez que isso pode ocorrer se não houver um gerenciamento dos processos, sendo este também um fator que muitas vezes causa certa resistência para a adoção deste recurso nos projetos.

Outra tecnologia extremamente importante que pode ser utilizada em conjunto com os Streams ou quaisquer outros recursos são as Threads. Elas permitem um melhor aproveitamento do processamento, uma vez que possibilitam a execução paralela de tarefas distintas ou semelhantes, oferecendo também a base para o funcionamento de uma ferramenta de Download segmentada e permitindo o desenvolvimento de aplicações MultiThread.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Maiores informações sobre Streams no Delphi
http://docwiki.embarcadero.com/Libraries/en/System.Classes.TStream

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Conhecimento faz diferença!

Faça já sua assinatura digital ! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional.
Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software.
Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviçoes); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia

Desenvolvendo para Android – Parte 4

RadPHP e Delphi XE 2

Este artigo faz parte de um curso

Resumo DevMan

De que se trata o artigo:

Esta quarta e última parte deste minicurso tem o objetivo de concluir o desenvolvimento do Servidor de Aplicação DataSnap e o Software Cliente (Web Application) que será exportado para a plataforma Android, utilizando as ferramentas Delphi XE 2 e RadPHP XE 2, tendo seu desenvolvimento baseado no Framework jQuery.

Em que situação o tema é útil:

Elaborar rapidamente um simples Software para a plataforma Web e Android empregando tecnologias WebStandards (HTML, CSS e Java Script), apresentando formas de se beneficiar com os vários recursos dos Frameworks jQuery e jQM (jQueryMobile).

Desenvolvendo para Android :

Em meio à grande e ascendente popularidade dos dispositivos móveis, a criação de aplicações para tais dispositivos se torna uma ótima opção para os desenvolvedores que querem inovar e entrar para um novo nicho de mercado. O RAD Studio XE 2 conta com uma gana de ferramentas que permitem a elaboração de aplicações para os mais variados Sistemas Operacionais, assim como o Android, através do RadPHP. Nesta última parte deste minicurso, o servidor DataSnap e a aplicação cliente apresentados anteriormente, receberão novos métodos e ambos os Softwares serão finalizados. Para concluir, o projeto será exportado para o emulador ou um dispositivo real baseado no sistema Android, através do assistente de exportação que utiliza a API PhoneGap, que por sua vez é compatível com diversos sistemas operacionais.

Nos artigos anteriores deste minicurso foram abordados todos os recursos necessários para o desenvolvimento de um conjunto de aplicações para prover funcionalidades para a plataforma móvel baseada em Android, tais como o Servidor de Aplicação DataSnap, as tecnologias REST (Representational State Transfer), HTML (HyperText Markup Language), CSS (Cascading Style Sheets), JavaScript e a PhoneGap. Nesta etapa final da aplicação móvel, iniciada no terceiro artigo, serão necessárias as inclusões de dois novos métodos para o Servidor de Aplicação DataSnap com o intuito de disponibilizar remotamente os serviços restantes que ainda não foram implementados, tais como a obtenção dos itens e a efetivação de uma venda, bem como novas funções na aplicação móvel Android, elaborada com a ferramenta RadPHP XE 2 e a API PhoneGap, que por sua vez permite o emprego das tecnologias padrões da Web (WebStandards) para o desenvolvimento de aplicações híbridas nativas destinadas a plataforma móvel.

Como também mencionado nos artigos anteriores, os Frameworks jQuery e jQM (jQueryMobile) foram adotados como a base para o funcionamento de toda a Aplicação Cliente deste projeto. Vale à pena lembrar que a jQuery é uma biblioteca livre, composta por uma diversidade de funções que agilizam e simplificam o desenvolvimento de documentos HTML e consequentemente, a elaboração de aplicações destinadas ao ambiente Web. Com relação à jQM, ela é uma extensão da jQuery que tem como alvo a plataforma móvel, contendo outras funcionalidades mais especificas, assim como componentes visuais característicos desta plataforma.

Tutorial

Novos métodos para o Servidor Datasnap

Em ambientes heterogêneos, um importante recurso que pode ser empregado para a centralização das regras de negócio da organização é o Servidor de Aplicação DataSnap. Através deste, a instituição

pode disponibilizar para seus colaboradores os mais variados e completos serviços localmente ou de forma remota, caracterizando o conceito de Software Cliente-Servidor. O desenvolvedor elabora aplicações multicamadas e, com o emprego da tecnologia REST (Representational State Transfer), pode facilmente criar sistemas para outras plataformas, como é o caso do ambiente móvel. O REST pode ser definido como um conceito ou uma arquitetura que é baseada em padrões presentes na Web, como é o caso do protocolo HTTP, o que também implica que uma aplicação elaborada dentro deste contexto é compatível com uma diversidade de Sistemas Operacionais e Softwares que suportem tal protocolo.

Para dar continuidade ao Servidor Datasnap, o projeto criado anteriormente neste minicurso deve ser reaberto para a adição de duas novas funções. A primeira que será adicionada, tem o intuito de obter todos os itens, ou aqueles cuja descrição contém parte do parâmetro informado, utilizando como base a tabela denominada Item, apresentada na segunda parte deste minicurso. Portanto, o módulo denominado ServerMethods1 deve ser selecionado. Este módulo é derivado da classe TDSServerModule, que por sua vez permite a exposição dos métodos e conjuntos de dados do Servidor para as aplicações clientes e foi automaticamente criado quando o servidor DataSnap foi concebido. Para a função mencionada, um objeto derivado da classe TDBXCommand deve ser empregado. Esta classe, integrante do Framework dbExpress, permite que instruções SQL ou procedimentos armazenados possam ser executados, onde a classe TDBXCommandTypes também deve ser utilizada para que o tipo do comando possa ser definido, salientando que o mesmo deve trabalhar em conjunto com uma conexão (TSQLConnection). Outro ponto importante é que um objeto do tipo TDBXCommand tem seu valor padrão retornado do tipo TDBXReader, que por sua vez é um leitor unidirecional para os registros de uma tabela do banco de dados. A **Listagem 1** apresenta o código do método GetItens que é baseado no objeto TDBXCommand.

Nesta função, o resultado retornado é do tipo TDBXReader, que dentro deste contexto, é automaticamente convertido em um documento baseado na notação JSON (JavaScript Object Notation), lembrando que tal notação é o padrão para a definição de objetos na linguagem JavaScript e é adotado neste minicurso para a transferência de informações entre o Servidor e a Aplicação Cliente através do protocolo HTTP. Posteriormente, é efetuado um comando Trim no parâmetro Descricao para limpar espaços desnecessários e verificar se algo foi preenchido, o que implica na elaboração de um filtro baseado na instrução SQL LIKE. Em seguida, o comando é criado para o objeto de conexão SQLConnection1 através da função DBXConnection.CreateCommand e seu tipo é definido como SQL, conforme a utilização de TDBXCommandTypes.DbxSQL. A propriedade Text do objeto TDBXCommand é atribuída com todas as instruções SQL para que a consulta de itens possa ser efetuada. Também é verificado se o objeto está pronto (através de .IsPrepared) e, caso não esteja, o mesmo é preparado através do método Prepare. Para concluir, é efetuada a execução das instruções SQL através de ExecuteQuery e o resultado é atribuído para o retorno da função.

Antes de dar início à próxima função, que por sua vez será responsável por efetuar a venda através do dispositivo móvel, as tabelas de venda e itens da venda devem ser criadas. A **Listagem 2** a presenta o código SQL que foi utilizado para a criação da tabela de vendas e a **Listagem 3** demonstra o código para a tabela de itens da venda.

Listagem 1. Método GetItens

```
01 function TServerMethods1.GetItens(Descricao: string): TDBXReader;
02 var
03   Comando: TDBXCommand;
04   Filtro: string;
05 begin
06   if Trim(Descricao) <> '' then
07     Filtro := ' AND DESCRICAO LIKE '+ QuotedStr('%'+ Descricao + '%');
08   Comando := SQLConnection1.DBXConnection.CreateCommand;
09   Comando.CommandType := TDBXCommandTypes.DbxSQL;
10   Comando.Text := ' SELECT CODIGO, DESCRICAO, VALORVENDA,
       QUANTIDADE '+
11   ' FROM ITEM WHERE ATIVO = "S"'+
12     Filtro +' ORDER BY CODIGO ';
13   if not Comando.IsPrepared then
14     Comando.Prepare;
15   Result := Comando.ExecuteQuery;
16 end;
```

Listagem 2. Código SQL da tabela VENDA

```
CREATE TABLE VENDA (
 CODIGO              INTEGER NOT NULL,
 CODIGOCLIENTE   INTEGER,
 DATA                 DATE,
 VENDEDOR          VARCHAR(30),
 TOTALVENDA       NUMERIC(9,2),
 FORMAPGTO        VARCHAR(5),
 EFETIVADA          VARCHAR(1)
);
```

Listagem 3. Código SQL da tabela VENDAITENS

```
CREATE TABLE VENDAITENS (
  CODIGOVENDA  INTEGER NOT NULL,
  CODIGOITEM     INTEGER NOT NULL,
  QUANTIDADE     NUMERIC(9,3),
  VALORVENDA     NUMERIC(9,2),
  SUBTOTAL        NUMERIC(9,2)
);
ALTER TABLE VENDAITENS ADD CONSTRAINT PKY_VENDAITENS
 PRIMARY KEY (CODIGOVENDA, CODIGOITEM);
ALTER TABLE VENDAITENS ADD CONSTRAINT FKY_VI_ITEM
 FOREIGN KEY (CODIGOITEM) REFERENCES
  ITEM (CODIGO) ON UPDATE CASCADE;
ALTER TABLE VENDAITENS ADD CONSTRAINT FKY_VI_VENDA
  FOREIGN KEY (CODIGOVENDA) REFERENCES VENDA (CODIGO)
  ON UPDATE CASCADE;
```

Com as tabelas devidamente criadas, a próxima etapa é elaborar a função responsável pela venda no servidor de aplicação. A **Listagem 4** apresenta o código da função InserirVenda, que por sua vez, deve ser adicionada ao módulo ServerMethods1 do projeto.

Nesta função, o resultado retornado é do tipo Boolean, indicando se a venda foi efetuada corretamente. Em um primeiro momento, é criada uma variável do tipo TSQLQuery, uma classe

do Framework dbExpress para efetuar comandos baseados em SQL no banco de dados, tais como consultas. Também é declarada uma variável do tipo TDBXTransaction. Esta última classe permite definir o início de uma transação, seu tipo e consequentemente, confirmar todo o processo (Commit) ou possibilita o cancelamento da operação (Rollback). Vale à pena salientar que uma transação é um processo de tudo ou nada, onde todos os comandos contidos entre o início e o fim da mesma devem ser executados de forma satisfatória ou nenhum deve ser efetuado, desfazendo as alterações até então. Declarada todas as variáveis, a classe TSQLQuery é instanciada. A variável CodigoVenda tem seu valor definido através da função GetNovoCodigo, apresentada na segunda parte deste minicurso, cujo o intuito é simplesmente obter o último valor inteiro de determinado campo de uma tabela (MAX) e incrementá-lo, gerando assim um novo código sequencial para a chave primária. Uma vez que várias instruções SQL serão executadas em série, a transação é iniciada através do comando SQLConnection1.BeginTransaction, onde seu tipo é passado como parâmetro (TDBXIsolations.ReadCommitted) conforme pode ser visto na Linha 10.

Posteriormente, são definidas as propriedades do objeto TSQLQuery (Qry), como a conexão e o primeiro comando SQL que será executado através da propriedade Text, que neste processo tem o objetivo de obter o valor do item que foi vendido, uma vez que não será permitida a alteração do valor do mesmo pela aplicação cliente para este exemplo. É possível observar que a função IntToStr é utilizada para converter um inteiro para String já que o comando SQL deve ser informado como texto. Obtido o valor do item vendido com a abertura da query (Open) e seu armazenamento em uma variável, é calculado o total da venda de acordo com a quantidade informada (Linha 15). Posteriormente, o mesmo objeto (Qry) é utilizado para gravar a venda no banco de dados, através da instrução SQL de INSERT (Linha 21). Em conjunto com o mesmo são utilizadas algumas funções importantes para auxiliar neste processo. A primeira é a QuotedStr, cujo objetivo é adicionar os caracteres de aspa simples (‘’) na String informada, essenciais para a manipulação deste tipo de dado no banco. A função FormatDateTime também é utilizada para alterar o formato da data (yyyy-mm-dd / ano-mes-dia) para o formato correto a ser gravado no banco de dados, tendo seu valor retornado como String. Para concluir, também é empregada a função StringReplace, que substitui determinados caracteres por outros, para a formatação e gravação de um campo do tipo Float, uma vez que o mesmo foi convertido em String. A instrução SQL montada com estas características concentra-se na Linha 21. Continuando a explanação da listagem, a próxima etapa é gravar o item vendido de acordo com a venda, na tabela VendaItens, onde o código da venda é informado e a função StringReplace é novamente utilizada. Desta vez é criada uma instrução SQL de UPDATE na tabela de itens com o intuito de atualizar o estoque, uma vez que o item tenha sido

Listagem 4. Método InserirVenda

```
01 function TServerMethods1.InserirVenda(Pessoa, Item: Integer;
   Qtde: Double;out Mensagem: string): Boolean;
02 var
03   Qry: TSQLQuery;
04   TotalVenda, ValorItem: Double;
05   Tran: TDBXTransaction;
06   CodigoVenda: Integer;
07 begin
08   Qry := TSQLQuery.Create(nil);
09   CodigoVenda := GetNovoCodigo('VENDA','CODIGO');
10   Tran := SQLConnection1.BeginTransaction(TDBXIsolations.ReadCommitted);
11   try
12     with Qry do
13     begin
14       SQLConnection := SQLConnection1;
     //Obtem valor do item
15       SQL.Text := ' SELECT VALORVENDA FROM ITEM WHERE CODIGO = '+
             IntToStr(Item);
16       Open;
17       ValorItem := FieldByName('VALORVENDA').AsFloat;
18       TotalVenda := ValorItem * Qtde;
19       Close;
20       SQL.Clear;
     //Efetua a venda
21       SQL.Text := ' INSERT INTO VENDA '+
             ' VALUES ('+ IntToStr(CodigoVenda) +','+
             IntToStr(Pessoa) +','+
             QuotedStr(FormatDateTime('yyyy-mm-dd', Date)) +','+
             QuotedStr('MOBILE') +','+
             StringReplace(FloatToStr(TotalVenda),',','.',[]) +',
             '+QuotedStr('VISTA') + ','"S"' +');';
22       ExecSQL;
23       Close;
24       SQL.Clear;
     //Insere o item da venda
25       SQL.Text := ' INSERT INTO VENDAITENS '+
             ' VALUES ('+ IntToStr(CodigoVenda) +','+
             IntToStr(Item) +','+
             StringReplace(FloatToStr(Qtde),',','.',[]) +','+
             StringReplace(FloatToStr(ValorItem),',','.',[]) +',
             '+ StringReplace(FloatToStr(ValorItem * Qtde),',','.',[]) +');';
26       ExecSQL;
27       Close;
28       SQL.Clear;
     //Atualiza o estoque
29       SQL.Text := ' UPDATE ITEM SET QUANTIDADE = QUANTIDADE - '+
             StringReplace(FloatToStr(Qtde),',','.',[]) +
             ' WHERE CODIGO = '+ IntToStr(Item);
30       ExecSQL;
31       Close;
32     end;
33     SQLConnection1.CommitFreeAndNil(Tran);
34     Mensagem := 'Processo Concluído!';
35     Result := True;
36   except
37     on e: Exception do
38     begin
39       SQLConnection1.RollbackFreeAndNil(Tran);
40       Mensagem := 'Houve um erro: ' + e.Message;
41       Result := False;
42     end;
43   end;
44   FreeAndNil(Qry);
45 end;
```

vendido (Linha 29). Para concluir, se todos estes processos foram realizados corretamente, a transação é confirmada e liberada da memória através do método CommitFreeAndNil tornando assim o resultado definido como verdadeiro, caso contrário, ela é desfeita por meio da método RollbackFreeAndNil e o retorno da função é definido como falso. O objeto TSQLQuery (Qry) também é liberado da memória (Linhas 33 a 44)

Neste momento, a definição das funções do servidor de aplicação DataSnap para este simples exemplo está finalizada.

Concluindo a aplicação cliente

Uma vez que existem novos métodos no Servidor de Aplicação, a aplicação Cliente para a plataforma Android também será atualizada. Para dar continuidade ao mesmo, o projeto apresentado na terceira parte deste minicurso deve ser reaberto, lembrando que o mesmo foi desenvolvido com a IDE do RadPHP XE 2, que por sua vez, utiliza a linguagem de programação PHP como base para o desenvolvimento de WebApps que podem ser facilmente exportadas para outros ambientes.

O primeiro passo para a criação da venda através da aplicação cliente é atualizar a tela (painel) de menus, elaborado anteriormente, adicionando um novo botão do tipo MButton, presente na aba jQueryMobile, que neste exemplo foi denominado btnIniciarVenda. A **Figura 1** demonstra a aparência final do painel de menus após a adição do botão para a venda.

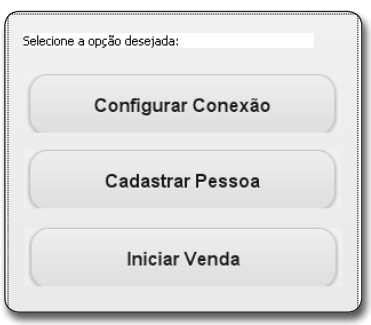


Figura 1. Painel referente ao Menu

Após a atualização da tela de menu, o próximo passo é criar duas novas telas (painéis) para a realização da venda. A Primeira tem o intuito de efetuar uma filtragem inicial para as informações que farão parte da venda, minimizando assim o tamanho da informação que será enviada como retorno, e a segunda, com o objetivo de permitir ao vendedor selecionar a pessoa e o item, informando também a quantidade, bem como a possibilidade de confirmar ou cancelar a venda. No primeiro painel foram inseridos 3 Labels, 2 MEdits e 1 MButton. A **Figura 2** apresenta o visual final do painel da tela de filtragem.

O painel referente a segunda tela terá dois componentes do tipo MComboBox, presente na aba jQueryMobile, que por sua vez também possui um visual característico do ambiente móvel.

Um ponto importante que deve ser ressaltado é com relação ao documento HTML que será gerado, onde este MComboBox é definido através das tags <select>[opções]</select>, sendo que cada opção deve estar contida em tags <option></option>. O conhecimento destas tags é necessário para que os itens de ambos os objetos possam ser adicionados facilmente através da linguagem JavaScript. Também foram inseridos 4 Labels, 2 MButtons e 1 MEdit, para que a quantidade seja informada. A **Figura 3** apresenta o visual final do painel da tela de vendas.

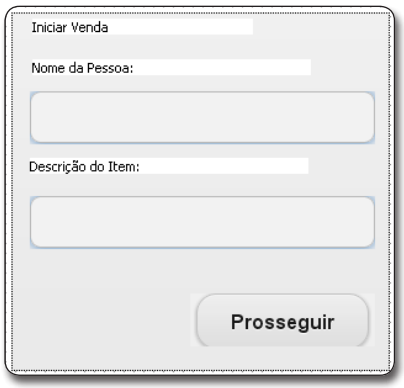


Figura 2. Painel referente a pré-filtragem das informações

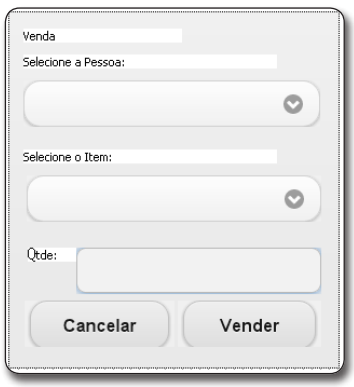


Figura 3. Painel referente a tela de confirmação da venda

Elaboradas as telas necessárias para que a venda seja efetuada corretamente, será criado um novo objeto através da linguagem JavaScript utilizando Object Literals, para auxiliar na gravação das informações da venda atual. Este recurso permite a concepção de objetos, suas propriedades e valores de uma forma bem simplificada. A **Listagem 5** apresenta o código do objeto denominado "vendaatual", que por sua vez, deve ser inserido logo após o objeto criado anteriormente para este projeto, denominado "conexao", presente na Tag HTML <script>.

Ao analisar o código é possível notar que as propriedades do mesmo servem para armazenar os dados da pessoa e do item atual que será vendido através da Aplicação Cliente, auxiliando assim o processo da venda.

A próxima etapa é iniciar o desenvolvimento dos códigos dos novos métodos para as telas e objetos criados anteriormente. Como o Framework jQuery foi utilizado como o "motor" principal deste projeto, todos os códigos à partir deste momento devem ser

inseridos no bloco criado no artigo anterior, jQuery(document).ready, para o correto funcionamento, uma vez que este código é executado e monitorado quando o documento HTML é devidamente carregado, ou também utilizando o método da jQM que trabalha de forma semelhante, jQuery(document).bind("mobileinit", function($){...}). O primeiro botão que teve seu evento "click" definido foi o btnIniciarVenda, cujo intuito é apenas carregar a tela de pré-filtragem das informações. A **Listagem 6** demonstra o código responsável pela manipulação do evento click do botão "btnIniciarVenda".

Listagem 5. Declaração do objeto vendaatual

```
var vendaatual = {
   pessoacodigo: '',
   pessoanome: '',
   itemcodigo: '',
   itemdescricao: '',
   itemvalor: ''
}
```

Listagem 6. Código do evento click do botão btnIniciarVenda

```
$('#btnIniciarVenda').click(function() {
   Mudar($('#mpMenu'),$('#mpIniciarVenda'));
   return false;
});
```

Em um documento HTML é uma boa prática definir a propriedade "id" para cada elemento que faz parte do mesmo com o intuito de informar um identificador para tal objeto, facilitando o acesso direto a ele através da linguagem JavaScript por exemplo, entretanto, existem outras formas de acessar tal componente sem ser por esta propriedade, utilizando o array de elementos que é criado automaticamente para os objetos. Como o Framework adotado para este projeto é a jQuery, um elemento pode ser facilmente acessado pela sua propriedade "id" simplesmente ao utilizar o comando $(''), como é o caso do botão deste trecho de código, $('btnIniciarVenda'), porém o valor definido para a identificação do objeto deve ser precedido do caractere "#". Também é possível notar que o evento click para o mesmo é manipulado, cuja função neste trecho é chamar o método criado anteriormente, "Mudar", para que a troca entre as telas possa ser realizada, lembrando que o primeiro parâmetro obtido através do id, mpMenu, é o painel que será ocultado, e o segundo, mpIniciarVenda, é o painel que será exibido. Vale a pena ressaltar que a jQM também possui uma forma de definir várias subpáginas em apenas uma, através do atributo data-role, possuindo inclusive animações, porém, que não foram adotadas para este exemplo. Para concluir, o método return false é utilizado para que não seja efetuada a ação padrão que é adotada quando um clique em um botão é realizado, uma vez que isso poderia ocasionar o recarregamento do documento HTML.

O próximo botão que terá seu código desenvolvido é o btnProsseguir, cujo intuito é ajudar na filtragem das informações e preencher os objetos que fazem parte da tela de vendas com os dados retornados pelo servidor de aplicação. A **Listagem 7** apresenta o código do botão "btnProsseguir", presente no painel "mpInserirVenda".

Listagem 7. Código do evento click do botão btnProsseguir

```
01 $('#btnProsseguir').click(function() {
02       var Parametro = $('#edtPesNome').val();
03       var Parametro2 = $('#edtPesItem').val();
04       Parametro = Parametro.toUpperCase() +'/';
05       Parametro2 = Parametro2.toUpperCase() +'/';
06       var Uri = 'http://' + conexao.ip + ':' + conexao.porta +
07             conexao.path + conexao.servicos[1] + Parametro;
08       var UriItens = 'http://' + conexao.ip + ':' + conexao.porta +
09             conexao.path + conexao.servicos[3] + Parametro2;
10       $.getJSON(Uri, function(pessoas) {
11         $('#cbxPessoas').empty();
12         $.each(pessoas.result[0].CODIGO, function(pos, codigo) {
13           $('#cbxPessoas').append('<option value="'+
14                 pessoas.result[0].CODIGO[pos] + '">' +
15                 pessoas.result[0].CODIGO[pos] + '-' +
16                 pessoas.result[0].NOME[pos] + '</option>');
17         });
18         $('#cbxPessoas').selectmenu('refresh');
19       });
20       $.getJSON(UriItens, function(itens) {
21         $('#cbxItens').empty();
22         $.each(itens.result[0].CODIGO, function(pos, codigo) {
23           $('#cbxItens').append('<option value="' +
24                 itens.result[0].CODIGO[pos] +'">'+
25                 itens.result[0].CODIGO[pos] + '-' +
26                 itens.result[0].DESCRICAO[pos] + '-' +
27                 itens.result[0].VALORVENDA[pos] + '</option>');
28         });
29         $('#cbxItens').selectmenu('refresh');
30       });
31       $('#cbxPessoas').change(function() {
32         vendaatual.pessoacodigo = $('#cbxPessoas').val();
33       });
34       $('#cbxItens').change(function() {
35         vendaatual.itemcodigo = $('#cbxItens').val();
36       });
37       $('#edtQtde').val('');
38       Mudar($('#mpIniciarVenda'),$('#mpVenda'));
39       return false;
40     });
```

No início deste trecho de código, nas linhas 1 e 2, são criadas duas variáveis, Parametro e Parametro2, que recebem os valores presentes nos respectivos objetos do tipo MEdit, utilizando os identificadores dos mesmos (id) e os recursos do Framework jQuery, assim como o método val, cuja função é obter ou atribuir um valor a determinado objeto. Em seguida, através da função toUpperCase, as informações das variáveis são convertidas para maiúsculo. Posteriormente, são criadas duas novas variáveis, Uri e UriItens (linhas 6 e 8), cujo objetivo é a elaboração das URIs que serão utilizadas para o acesso ao servidor de aplicação, obtendo também as informações que foram definidas anteriormente com o objeto "conexao". O Framework jQuery possui diversos métodos úteis e que podem ser utilizados rapidamente, como é o caso do get JSON. Esta função se assemelha ao método ajax deste framework. Ela utiliza o HTTP GET e é destinada a obtenção de documentos baseados na notação JSON, podendo criar automaticamente

o objeto, onde o primeiro parâmetro é o endereço (URL ou URI) que deve ser acessado, e o segundo, uma função que indica o processo que será realizado, caso a mesma seja executada satisfatoriamente, que por sua vez tem como parâmetro o objeto criado com base no retorno, que foi denominado "pessoas" neste exemplo. Em seguida, no início do bloco desta função, é efetuada uma limpeza nas informações contidas no MComboBox cbxPessoas através do método empty (linha 11). Ao utilizar um retorno do tipo TDBXReader no Servidor de Aplicação DataSnap para uma função, o Delphi cria um documento baseado em JSON para a resposta contendo uma estrutura característica e pré-definida, e após a conversão do mesmo para um objeto em JavaScript, ele poderá ser acessado através da expressão: "<nome do objeto>.result[0].<nome do campo da tabela>[posição]". Dentro deste contexto, a função each (linha 12) é utilizada para executar o trecho de código seguinte para cada ocorrência do campo Código, presente no objeto denominado pessoas. Sendo assim, o MComboBox (cbxPessoas) adiciona através do método append (linha 13), o código da pessoa com o seu valor padrão (propriedade Value do elemento <option>), e exibe tal código e o nome para a descrição da pessoa e, ao finalizar o método each, ele recebe uma atualização, uma vez que os itens foram recarregados, sendo a mesma efetuada através do método selectmenu acompanhado do parâmetro refresh (linha 18). Em seguida, na linha 20, o mesmo método getJSON é chamado novamente, porém agora para obter os itens ou produtos. No objeto cbxItens, que por sua vez receberá as informações retornadas, é utilizado empty para a limpeza dos valores presentes. Também é utilizado o each sob os dados retornados, adicionando assim ao objeto cbxItens o código, a descrição e o valor para cada item retornado. Concluída esta etapa, o MComboBox também é atualizado para a correta visualização das novas informações através do comando selectmenu seguido do parâmetro refresh.

Continuando a explanação da listagem é definida a ação para o evento change (linha 31), executado quando o valor é alterado, para o MComboBox (cbxPessoas), para que a pessoa selecionada seja armazenada no objeto criado anteriormente, "vendaatual", com a utilização do método val. Da mesma forma, o objeto cbxItens tem seu evento change definido para a gravação do item que foi selecionado, onde o objeto "vendaatual" também recebe as informações do mesmo. Para concluir, o objeto edtQtde, que é do tipo MEdit, tem seu valor eliminado através do método val e é efetuada a troca da tela ou painel através do método Mudar, chamando assim a tela de vendas (mpVenda) e retornando o valor como falso para que nenhuma ação padrão seja efetuada para o evento de clique.

Terminado o código do botão btnProsseguir, o próximo passo é definir o código para o botão de cancelamento de uma venda, btnCancelarVenda. A **Listagem 8** demonstra o código para o cancelamento da venda.

Este trecho de código também tem apenas a função de chamar outra tela, neste caso o painel que contém o menu (mpMenu), saindo assim da tela de vendas com a utilização do método Mudar e retornando falso para a ação padrão.

Para concluir, o último código que deve ser inserido é o responsável por efetuar a venda junto ao Servidor de Aplicação DataSnap. A **Listagem 9** apresenta o código do evento click do botão btnVender, que por sua vez, confirma a venda realizada.

Listagem 8. Código do evento click do botão btnCancelarVenda

```
$('#btnCancelarVenda').click(function() {
  Mudar($('#mpVenda'),$('#mpMenu'));
  return false;
});
```

Listagem 9. Código do evento click do botão #btnVender

```
01 $('#btnVender').click(function() {
02       var CodigoPessoa = vendaatual.pessoacodigo + '/';
03       var CodigoItem = vendaatual.itemcodigo + '/';
04       var Qtde = $('#edtQtde').val() + '/';
05       var UriVenda = 'http://' + conexao.ip + ':' + conexao.porta +
06                conexao.path + conexao.servicos[4] +
07                CodigoPessoa + CodigoItem + Qtde;
08       alert(UriVenda);
09       $.ajax({
10         type: 'GET',
11         url: UriVenda,
12         contentType: 'application/json; charset=utf-8',
13         dataType: 'json',
14         success: function (data) {
15           $.each(data, function (i, retorno) {
16             alert(retorno.toString());
17           });
18         },
19         error: function () {
20           alert('Houve um erro!');
21         }
22       });
23       Mudar($('#mpVenda'),$('#mpMenu'));
24       return false;
25     });
```

No início deste trecho de código, são criadas e definidas duas variáveis (linhas 2 e 3), CodigoPessoa e CodigoItem, para o armazenamento das informações necessárias para a venda, presentes no objeto "vendaatual". Também é criada na linha 3 uma variável para armazenar a quantidade que será vendida, onde o valor é atribuído de acordo como o preenchimento do campo edtQtde e com a utilização do método val. A URI para a gravação da venda também é montada com auxílio do objeto "conexão" seguido pelos valores das variáveis (linha 5), sendo o mesmo armazenado na variável UriVenda. Posteriormente na linha 9, o método ajax do jQuery é empregado para a requisição assíncrona dos dados. Para este método também são definidas as informações necessárias, tais como type, o tipo da forma de acesso (HTTP GET), a url para o endereço da requisição (UriVenda), o contentType e o dataType para informar o tipo da informação (JSON), o success, para definir a ação que será efetuada caso o processo seja concluído sem problemas, onde neste caso é exibida uma mensagem contendo o valor de retorno através do método alert, e o error, que por sua vez define qual a ação que deve ser tomada caso ocorra algum erro. Para concluir, após a mensagem, a tela de menus é chamada

novamente através do método Mudar (linha 23) e também é efetuado o return false.

Como pode ser observado, a Aplicação Cliente pode ser completamente e facilmente desenvolvida baseada nos recursos dos Frameworks jQuery e jQM. Outro ponto extremamente importante que deve ser observado é com relação aos navegadores que serão utilizados para o processo de Deploy da Aplicação Cliente durante o período de testes, pois os mesmos podem barrar o correto funcionamento dos métodos devido as suas configurações de segurança. O navegador utilizado para este projeto foi o Google Chrome, contudo o mesmo deve ser carregado com a segurança desabilitada para a Web. Para tanto, basta selecionar o atalho do navegador, clicar com o botão direto sobre o mesmo e acessar a opção Propriedades. O comando "disable-web-security" deve ser adicionado no final do campo Destino, que por sua vez contém o endereço do executável do Chrome, permitindo assim que todos os métodos apresentados funcionem corretamente.

Exportanto a aplicação cliente para o Android

Uma vez que a aplicação esteja com todas as funcionalidades definidas, bem como tenha sido testada e depurada em um navegador, é o momento de exportá-la para o sistema foco deste projeto, o Android. Vale a pena lembrar que o RadPHP utiliza a API PhoneGap para este processo, que por sua vez, é livre e permite a criação de aplicações híbridas nativas para diversas plataformas através de documentos HTML.

Para dar início a este processo, com o projeto aberto, basta ir até o menu Ferramentas e selecionar a opção Wizard For PhoneGap. A **Figura 4** demonstra a janela da ferramenta de exportação do projeto para outros ambientes.

Nesta janela basta selecionar a opção Android e prosseguir no botão Next do assistente. Posteriormente algumas informações como nome da aplicação e empresa devem ser definidas de acordo com o desejo do desenvolvedor. Após avançar, também pode ser selecionado um ícone para a aplicação. A próxima etapa é definir o local em que o projeto deve ser exportado. Neste exemplo o, diretório adotado foi "C:\ClienteAndroid\". Este local, que deve ser um endereço válido, conterá os arquivos necessários para a construção do pacote da aplicação para o Android (apk). Após avançar, o projeto é devidamente exportado, e na sequência, o desenvolvedor pode optar por criar a aplicação no emulador ou diretamente em um dispositivo físico devidamente instalado e conectado à porta USB. Vale ressaltar que quando um dispositivo físico é selecionado, tal como neste projeto que foi testado em um SmartPhone com o sistema operacional Android versão 2.2, a configuração denominada "Depuração de USB", presente em Configurações>Aplicações>Desenvolvimento no menu principal do aparelho, deve ser ativada. Concluída esta etapa, a aplicação é devidamente criada e enviada para o alvo selecionado e pode ser executada. A **Figura 5** apresenta a aplicação cliente DataSnap executando no emulador do sistema Android, com a tela de vendas aberta contendo alguns itens e pessoas cadastrados para os testes. Como foi possível observar, a exportação do projeto para o emulador ou um dispositivo físico é um processo simples e muito rápido através do assistente da ferramenta RadPHP XE 2.

Nota do DevMan

O URI (Uniform Resource Identifier – Identificador Uniforme de Recurso) é utilizado para identificar determinado recurso disponível em uma rede tal como a Internet, sendo composto pelo protocolo, como é o caso do HTTP, e uma sequencia de caracteres (String).

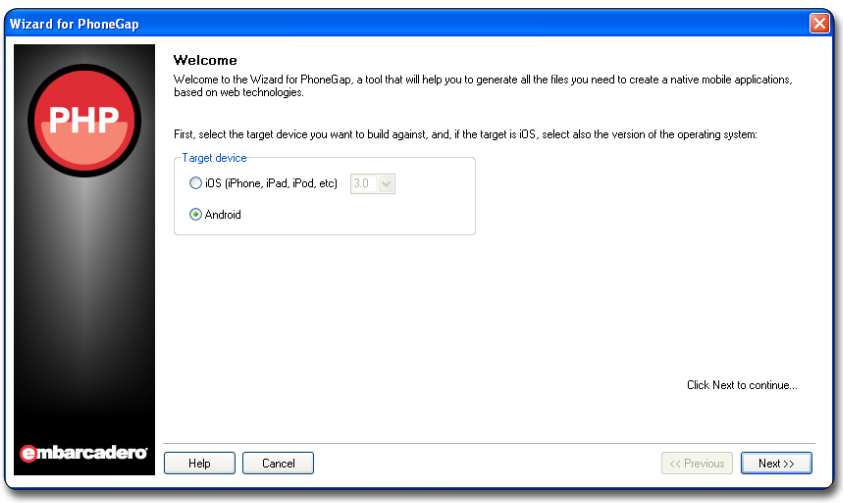


Figura 4. Janela do assistente de exportação, Wizard For PhoneGap

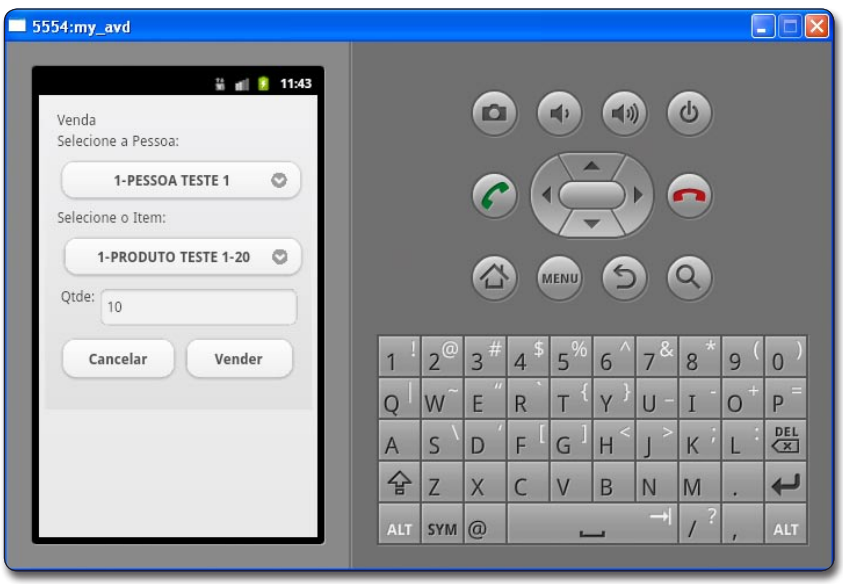


Figura 5. Aplicação Cliente executando no emulador do sistema Android

Conclusão

Com a ascendente popularização do ambiente móvel e de sistemas como Andorid, iOS, entre outros, o desenvolvimento para esta plataforma se tornou um diferencial competitivo e inovador para as Software Houses. Como foi possível notar, com as IDE's Delphi e RadPHP XE 2, o desenvolvedor pode criar um Servidor de Aplicação e um único Cliente para que seu Software seja compatível com qualquer Sistema Operacional que possua uma rede e suporte o protocolo HTTP, podendo executá-lo através de um navegador ou por meio de um dispositivo móvel com o auxílio da API PhoneGap.

Durante este minicurso a tecnologia DataSnap foi apresentada, bem como diversas linguagens do ambiente Web, como HTML e JavaScript, assim como dois dos principais Frameworks da atualidade para esta plataforma, a jQuery e a jQM.

Site oficial e manual do PHP
http://www.php.net/manual/pt_BR/

Maiores informações sobre JSON
http://www.json.org

Site oficial da biblioteca jQuery
http://www.jquery.com/

Maiores informações sobre o Framework jQueryMobile
http://www.jquerymobile.com/

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

CURSOS ONLINE

A Revista Clube Delphi oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

Clube Delphi

CONHEÇA ALGUNS DOS CURSOS:

- Rad Studio 2007-DBX4 e Firebird 2.0
  Aplicação comercial de vendas(básico) do Inicio ao Fim
- Dominando a Orientação a Objetos e Componentes em Delphi (Avançado)
- Novidades no Delphi 2010
- Curso Loja Virtual em Delphi Prism
- Aplicação Financeira com Delphi 2009 e MySQL

Para mais informações :
www.devmedia.com.br/cursos/delphi
(21) 3382-5038

DevMedia

dbExpress de A a Z – Parte 3

Recursos avançados

Este artigo faz parte de um curso

O Framework dbExpress é a forma mais simples e transparente de acesso a dados que o Delphi possui, através dela já é possível executar diversos comandos desde o CRUD básico (inserção, atualização, exclusão e seleção – **ver Nota do DevMan 1**), até mesmo a manipulação de dados, execução de Procedures, manipulação de Views, e quaisquer outros recursos que o banco nos ofereça, além é claro da comunicação destes dados. Porém, um assunto bastante trabalhado, mas pouco debatido, são os grandes processamentos de dados dentro do Delphi, onde na maioria das vezes, possuímos o cenário em que existe uma grande massa de dados que precisa ser obtida de um ponto e movida para outro.

Nota do DevMan 1

CRUD acrônimo de Create, Read, Update e Delete em língua Inglesa para as quatro operações básicas utilizadas em bancos de dados relacionais ou em interface para usuários para criação, consulta, atualização e destruição de dados.

Outra questão refere-se ao tratamento de Logs de forma simples, tanto em estruturas já existentes quanto as iniciadas. Destaque neste artigo para a forma simples e de baixo impacto em sistemas já consolidados. Além é claro, de uma questão importante que causa em diversos Softwares uma grande dor de cabeça quando se trata de manutenção, que é a abordagem da atualização de estruturas de dados em bancos já existentes.

É utilizando deste cenário como ponto de partida que este artigo começa. Serão abordadas maneiras de utilizar ao máximo os componentes dbExpress em benefício do desenvolvimento produtivo, visando sempre o código limpo, funcional e simples.

Resumo DevMan

De que se trata o artigo:

Na terceira parte deste curso demonstraremos os recursos avançados introduzidos no DBX Framework. Estes recursos permitem manipular facilmente informações de estrutura, permitindo utilizar o próprio Framework para a construção de outras soluções que podem inclusive derivar ferramentas de ORM (Object Relational Mapping).

Em que situação o tema é útil:

Manipular uma grande quantidade de dados através do Delphi, manter registro de todas as informações processadas, tanto em arquiteturas simples ou complexas, abrangendo situações como uma camada, múltiplas camadas e vários usuários. Além de obtermos aqui uma estrutura para contemplar o processo de manipulação da camada DDL nos bancos de dados, camada esta, que pode ser trabalhada sem a necessidade de utilização de comandos SQL, bastando apenas utilizar classes contidas dentro do Delphi. Todos estes processos serão bastante úteis em algumas tarefas como importações de dados ou atualizações de bancos. Além de trazer alguns conceitos para melhorar a codificação diária de qualquer sistema que utilize banco de dados relacional. Este artigo irá unir de forma simples como melhor utilizar os componentes de manipulação de dados de forma mais elegante, eficiente e simples.

dbExpress de A a Z – Parte 3 :

Nesta série de artigos iremos abordar a os recursos avançados do dbExpress responsáveis pela manipulação de grade massa de dados. Parte esta que ao longo dos anos sofreu grandes transformações, tornando-se hoje um Framework indispensável para quem deseja criar aplicações robustas, capazes de grandes processamentos e com baixo consumo de recursos, além de integrar cada vez mais o Delphi com os mais diversos bancos de dados, tornando assim a experiência com o acesso a dados algo mais confortável para quem necessita utilizar desta integração no cotidiano.

As execuções complexas do Delphi requerem sempre uma boa arquitetura do código, isto vai desde a escolha e configuração dos componentes, forma como se trabalha os recursos do banco em conjunto com o Delphi, além de boas práticas de código. Para o

cenário de importação e/ou atualização de base de dados, sempre temos alguns pontos de "gargalo", tais como: lentidão, adaptação de comandos SQL para diferentes bancos de dados, utilização de processamento e recursos de Hardware entre outros.

O dbExpress além de ser um Framework que auxilia o desenvolvedor no processo de persistência, também é capaz de manipular estruturas de bancos. Este recurso está disponível desde a versão 2009 do Delphi.

Neste artigo iremos criar um banco de dados no SQLServer 2008 Express e no Firebird 2.5 utilizando o mesmo conjunto de classes. Após o processo de criação, trataremos a inclusão de registros dentro do SQLServer, para em seguida, realizar a transmissão de dados de um para outro através do Delphi, tendo assim a capacidade de exemplificar as melhores maneiras para a realização deste processo de manipulação de grande massa de dados.

Um problema existente quando necessitamos trabalhar com grandes quantidades de informação, tanto para inserções quanto para as outras operações, é a quantidade de vezes que necessitamos ir ao banco de dados, isto ao longo do processo de uma importação pode se tornar bastante caro.

Com isso, poderemos visualizar abordagens de otimização de estruturas para consultas ao banco, tornando o processo inteiro algo mais simples, independente do SGDB e número de informações a serem transitadas.

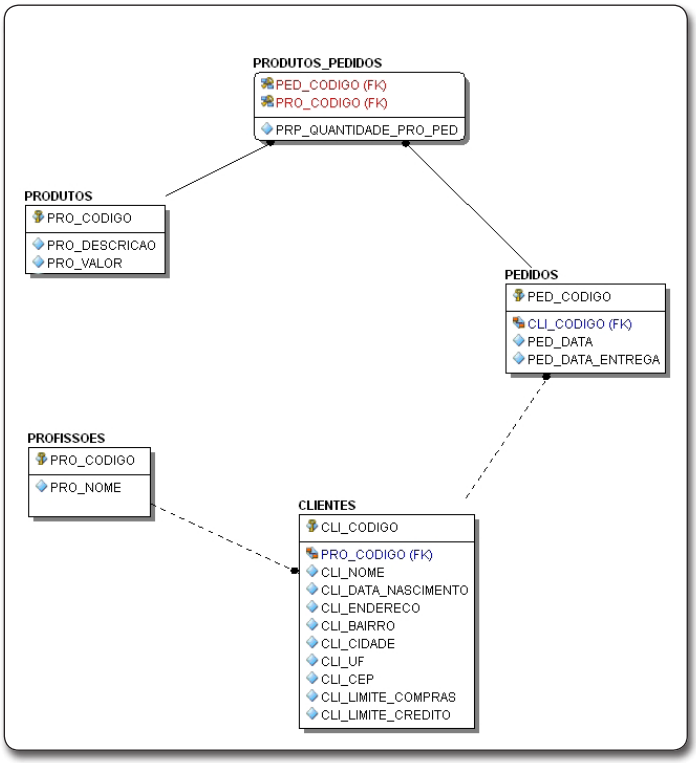


Figura 1. Modelo de dados

Introduzindo os recursos avançados

O dbExpress como conjunto de classes que estão contidas na unit (DbxCommon), além de trabalhar com os diversos bancos de dados já suportados no Delphi, também é a base para o trabalho com a característica de processo complexos, como a manipulação de grande quantidade de informações. Através de sua estrutura de classes que iremos criar tabelas em distintos bancos e manipular as informações de um para outro. Tudo isto utilizando de boas práticas de programação com o Delphi, visando o código limpo e utilizando o paradigma de orientação a objetos.

Utilizaremos uma estrutura de dados já trabalhada no primeiro artigo (**Figura 1**). Esta estará disponível para os bancos Firebird e SQL Server, sendo os dois exatamente idênticos, porém cada um com suas características particulares (tipos primitivos, sequências, triggers e etc).

Para começar, iremos criar estas tabelas sem a utilização de comandos SQL. Este é um recurso contido no dbExpress a partir da versão 4 em diante, na qual auxilia os desenvolvedores que necessitam realizar manutenção em diversos bancos.

Hoje, quando um só aplicativo tem a necessidade de trabalhar com diversos bancos simultaneamente, o processo de criação e atualização torna-se, na maioria das vezes, um ponto crítico que os desenvolvedores enfrentam, pois é parte vital desta tarefa, que nestes casos, é feita de forma manual e sem auxílio de outras ferramentas.

Como sabemos, os bancos de dados relacionais trabalham com uma estrutura de linguagem SQL ANSI que é comum entre eles, tomando isto como base, pode-se chegar a uma conclusão lógica. Para a manutenção podemos construir processos únicos e uniformes, que contemplem todas estas necessidades.

Uma forma de fazer esta tarefa é utilizando ferramentas de modelagem de dados, porém, para o profissional que se enquadra na figura de desenvolvedor que não participa da estruturação ou modelagem de um sistema. Estas opções, apesar de eficientes, não chegam a ser uma "solução final" para a situação.

Criando as tabelas

Para iniciar, crie uma nova aplicação VCL Forms (File>New>VCL Forms Application – Delphi), para que possamos interagir com o processo de criação. Dessa forma, desenvolva um layout semelhante ao da **Figura 2**.

Figura 2. Tela Principal

Os processos serão controlados por uma classe "assistente de criação", claro que para facilitar este exemplo, utilizaremos Generics (**ver Nota do DevMan 2**). Sendo assim, será feita a representação da estrutura do banco por meio de classes. Crie uma Unit nova (untTabela) e adicione o código da **Listagem 1** que

ilustra a representação da estrutura de classe referente as tabelas. Em seguida, em uma nova Unit (untCampo) adicione o código da **Listagem 2** na qual representa as colunas da tabela.

Listagem 1. Classe TTabela (representação de uma tabela)

```
//Adicione a unit untCampo (Listagem 2) à sessão uses
unit UntTabela;

interface

uses
 System.Generics.Collections,
 Data.SqlExpr,
 Data.DBXDataExpressMetaDataProvider,
 UntCampo;
type
 TTabela = class
 private
  FNome: String;
  FCampos: TList<TCampo>;
 protected
 public
  property Nome : String read FNome write FNome;
  property Campos : TList<TCampo> read FCampos write FCampos;
 end;
```

Note na classe TTabela (**Listagem 1**), que foi designada para ser a representação de uma tabela do banco de dados, a propriedade "Campos", que nada mais é do que uma lista genérica (Generics – **ver Nota do DevMan 2**). A adição da propriedade Campos como um "TList<TCampo>" sugere que teremos várias ocorrências de colunas por tabela, tratando assim, cada TCampo como um objeto de TTabela.

Nota do DevMan 2

Generics ou tipos genéricos descrevem listas de dados fortemente "tipados". Através deles é possível criar objetos próprios e representá-los de forma que não sejam necessários Type Castings. Em versões inferiores a 2009, qualquer tipo de lista de objeto deveria ser criada e validada por meio de Castings, o que de certa forma, não era uma forma segura, muitas vezes ocasionando erros de conversão.

Através de Generics não corremos esse risco, pois uma TList de objetos N só aceitará objetos do tipo N e nada mais, garantindo uma forma mais elegante e segura para a manipulação de objetos.

No código da **Listagem 2** temos a definição doa classe que armazenará oss campos. Primeiramente na linha 04, é possível perceber um tipo enumerado (ETipoCampo). Este tipo servirá para representar os tipos de dados permitidos. Observa-se a criação das propriedades da classe TCampo (linhas 12 a 16). Na linha 17 temos a declaração do método construtor. As propriedades criadas em TCampo são analogas ao que existem em. Exemplos das propriedades equivalentes Required, Size, Name e DataType (TField) e pfInKey disponível nos ProviderFlags.

Com o código da classe TCampo declarado, a seguir basta pressionar Ctrl + Shift + C que o Delphi se encarregará da implementação.

O método construtor (Create) deverá ser preenchido conforme o código da **Listagem 3**. O construtor simplesmente inicializa as variáveis de classe através dos parâmetros de entrada informados no momento da criação da classe TCampo.

O próximo passo é a elaboração da classe responsável por criar as tabelas em si. Sendo assim, crie uma nova Unit nomeando-a para untConstrutorTabela. Na **Listagem 4** temos a declaração da classe e a seguir, a explanação de seus respectivos métodos.

Listagem 2. Classe TCampo (representação das colunas de uma tabela)

```
01 unit untCampo;

02 interface

03 type
04  ETipoCampo = (tpString, tpInteger, tpDouble, tpDateTime);

05  TCampo = class
06  private
07   FNome: String;
08   FTipo: ETipoCampo;
09   FTamanho : Integer;
10   FRequirido : Boolean;
11   FChavePrimaria: Boolean;
12  public
13   property Nome : String read FNome write FNome;
14   property Tipo : ETipoCampo read FTipo write FTipo;
15   property Tamanho : Integer read FTamanho write FTamanho;
16   property Requirido : Boolean read FRequirido write FRequirido;
17   property ChavePrimaria: Boolean read FChavePrimaria write
     FChavePrimaria;
18   constructor Create(strNome : String; tpTipo : ETipoCampo;
                intTamanho : Integer = 0;
                boolRequerido : Boolean = False;
                boolChavePrimaria: Boolean = False);
19  end;
```

Listagem 3. Método construtor Create

```
01 constructor TCampo.Create(strNome: String; tpTipo: ETipoCampo;
    intTamanho: Integer; boolRequerido: Boolean; boolChavePrimaria:
    Boolean);
02 begin
03  FNome          := strNome;
04  FTipo          := tpTipo;
05  FTamanho       := intTamanho;
06  FRequirido     := boolRequerido;
07  FChavePrimaria := boolChavePrimaria;
08 end;
```

O código da **Listagem 5** ilustra o método ObterInstancia que utiliza a variável de classe FInstancia para armazenar uma instância da própria classe. Como se pode notar, ela implementa o padrão de projeto Singleton (**ver Nota do DevMan 3**).

Na **Listagem 5** a variável FInstancia que armazena instância da própria classe, é checada por meio do método Assigned, que retorna se o ponteiro para o objeto é nulo ou não. Caso a variável seja nula, é criada uma nova instância da classe e de qualquer forma, Result garante a existência de uma única classe sempre, retornando a própria instância criada ou existente.

Listagem 4. Classe TConstrutorTabela

```
01 unit UntConstrutorTabela;
02 interface
03 uses
     //Adicione untCampo e untTabela ao uses
    UntCampo,
    UntTabela,
    Data.SqlExpr,
    Data.DBXDataExpressMetaDataProvider;
04 type
05  TConstrutorTabela = class
06  private
07    class var FInstancia : TConstrutorTabela;
08   function ObterProvider(var Conexao : TSQLConnection) :
       TDBXDataExpressMetaDataProvider;
09    procedure AdicionarPrimaryKey(provider :
       TDBXDataExpressMetaDataProvider;Tabela : TTabela);
10   public
11    procedure CriarTabela(var Conexao: TSQLConnection;
12    var Tabela : TTabela; const Banco : Integer);
13    class function ObterInstancia : TConstrutorTabela;
14  end;
15 implementation
16 uses
    System.SysUtils,
    Data.DbxCommon,Data.DbxClient,
    Data.DbxMetaDataProvider;
```

Listagem 5. Método ObterInstancia da classe TConstrutorTabela

```
class function TConstrutorTabela.ObterInstancia: TConstrutorTabela;
begin
 if (not Assigned(FInstancia)) then
   FInstancia  := TConstrutorTabela.Create;
 Result  := FInstancia;
end;
```

Nota do DevMan 3

O padrão de projeto Singleton faz partes dos padrões Criacionais e tem por objetivo garantir a unicidade de um objeto. Neste caso, ela garante que uma única instância seja criada e retornada. É comumente utilizada para a representação de objetos de conexão. Geralmente este é um dos padrões de projeto mais utilizado pelos desenvolvedores Delphi.

Continuando a abordagem da classe TConstrutorTabela, a **Listagem 6** exibe a implementação do método ObterProvider.

No código da **Listagem 6** temos o seguinte: basicamente na linha 03 é declarada uma variável do tipo TDBXDataExpressMetaData Provider. Na linha 05 este objeto é criado, note que este objeto em si representa um Provider, que será algo essencial no processo de persistência. Na linha 07 o objeto Connection recebe um DBX-Connection, que é a representação da conexão em si (recebida por parâmetro através do objeto Conexao (TSQLConnection). Observe que na linha 08 o Provider é aberto e retornado a seguir na linha 13. Em caso de alguma exceção o objeto é destruído e a própria exceção é exposta através de raise (linha 11).

A próxima etapara a ser mostrada é a implementação do método AdicionarPrimaryKey, ilustrada na **Listagem 7**.

Listagem 6. Método ObterProvider da classe TConstrutorTabela

```
01 function TConstrutorTabela.ObterProvider(
   var Conexao: TSQLConnection): TDBXDataExpressMetaDataProvider;
02 var
03  Provider: TDBXDataExpressMetaDataProvider;
04 begin
05  Provider := TDBXDataExpressMetaDataProvider.Create;
06  try
07   Provider.Connection := Conexao.DBXConnection;
08   Provider.Open;
09   except
10    FreeAndNil(Provider);
11   raise;
12  end;
13  Result := Provider;
14 end;
```

Listagem 7. Método AdicionarPrimaryKey da classe TConstrutorTabela

```
01 procedure TConstrutorTabela.AdicionarPrimaryKey(
    provider: TDBXDataExpressMetaDataProvider; Tabela : TTabela);
02 var Indice : TDBXMetaDataIndex;
03    I: Integer;
04 begin
05  Indice := TDBXMetaDataIndex.Create;
06  try
07   Indice.TableName := Tabela.Nome;
08    for I := 0 to Tabela.Campos.Count -1 do
09     if (Tabela.Campos[I].ChavePrimaria) then
10      Indice.AddColumn(Tabela.Campos[I].Nome);
11   Provider.CreatePrimaryKey(Indice);
12  finally
13   FreeAndNil(Indice);
14  end;
15 end;
```

Note no código da **Listagem 7** a declaração de uma variável do tipo TDBXMetaDataIndex (linha 02). Como sabe-se, no contexto de bancos de dados relacionais cada chave cria um índice dentro do mesmo. A sugestão é que esta variável represente o índice referente a chave primária. Na linha 05 temos a criação do objeto Indice. A seguir na linha 07, a propriedade TableName do mesmo objeto (Indice) recebe o nome da tabela (passado por parâmetro a partir do objeto TTabela). Na linha 08 é realizado um laço de acordo com os campos existentes dentro do objeto Tabela. Observe que na linha 09 é checado campo a campo, verificando se este faz parte da chave primária. Caso seja identificado, ele é adicionado ao objeto Indice por meio do método AddColumn (linha 10). Por último, o Provider (também passado como parâmetro) recebe o índice, que é adicionado ao objeto (linha 11), sendo finalmente liberado da memória (linha 13).

Finalizando a classe TConstrutorTabela temos a **Listagem 8**, que introduz o método CriarTabela.

Finalizando a classe TConstrutorTabela, temos o código da **Listagem 8.** Entre as linhas 02 a 05 há a definição das variáveis utilizadas. No método CriarTabela temos 3 parâmetros de entrada. O primeiro denominado "Conexao" refere-se ao o objeto

TSQLConnection. O segundo, TTabela, representará a própria tabela a ser criada. E por último, "Banco", responsável por tornar nossa aplicação multi-banco. Na linha 07 é obtido um objeto Provider através do método ObterProvider (anteriormente criado na **Listagem 6**). A seguir na linha 08, é criado o objeto que representará a tabela (TabelaDbx). Assim como o código da **Listagem 7,** a tabela obtida por parâmetro é atribuída à propriedade TableName do objeto TabelaDbx (linha 10). Já na linha 11, é iniciado um laço "for" que percorrerá os campos contidos no objeto TTabela. Note que entre as linhas 13 a 19 são validados os tipos dos campos, mapeando-os conforme suas definições. Caso o banco de dados seja SQLServer, a proprieade AutoIncrement da coluna (específica deste banco) recebe o campo chave primaria, por fim, a coluna com suas definições é adicionada ao objeto TabelaDbx (linhas 20 a 22). Com isso é chamado o método CreateTable passando a tabela definida como parâmetro, respectivamente definindo a chave primária (linhas 24 e 25) e consequentemente destruindo e liberando os objetos da memória (linhas 27 e 28).

Com o código finalizado, basta apenas que a classe TConstrutorTabela seja chamada que o método CriarTabela se encarregará de manipular todo o processo restante. É possível notar também que nesta arquitetura não existe nenhum vínculo direto a nenhum banco de dados, pois a conexão é obtida através de parâmetro, suportando assim qualquer banco de dados cujo Delphi trabalha.

Listagem 8. Método CriarTabela da classe TConstrutorTabela

```
01 procedure TConstrutorTabela.CriarTabela(var Conexao: TSQLConnection;
     var Tabela: TTabela; const Banco : Integer);
02 var TabelaDbx : TDBXMetaDataTable;
03  Campo : TCampo;
04  Coluna : TDBXMetaDataColumn;
05  Provider : TDBXDataExpressMetaDataProvider;
06 begin
07  Provider := ObterProvider(Conexao);
08  TabelaDbx := TDBXMetaDataTable.Create;
09  try
10   TabelaDbx.TableName := Tabela.Nome;
11   for Campo in Tabela.Campos do
12   begin
13    case Campo.Tipo of
14     tpString : Coluna := TDBXUnicodeVarCharColumn.Create
      (Campo.Nome, Campo.Tamanho);
15     tpInteger : Coluna := TDBXInt32Column.Create(Campo.Nome);
16     tpDouble : Coluna := TDBXDoubleColumn.Create(Campo.Nome);
17     tpDateTime: Coluna := TDBXDateColumn.Create(Campo.Nome);
18    end;
19    Coluna.Nullable := not Campo.Requirido;
20    if (Banco = 0) then //Banco SQL Server
21     Coluna.AutoIncrement := Campo.ChavePrimaria;
22     TabelaDbx.AddColumn(Coluna);
23    end;
24    Provider.CreateTable(TabelaDbx);
25    AdicionarPrimaryKey(Provider, Tabela);
26  finally
27   FreeAndNil(TabelaDbx);
28   FreeAndNil(Provider);
29  end;
30 end;
```

Manipulando os dados

Após a criação da estrutura de dados, o processo seguinte se dá a manipulação de dados, onde é feita a transmissão das informações contidas no banco SQL Server para o Firebird de forma automática. O banco SQL Server foi preenchido com dados aleatórios gerados pela ferramenta "Generate Data" (disponível na sessão "**Links**" ao final do artigo). O processo de importação de dados em um aspecto geral não possui muitas divergências do que já trabalhamos no dia a dia, no entanto, o que é necessário um pouco mais de atenção são os pontos críticos que existem em arquiteturas e execuções.

Antes de iniciar qualquer tipo de importação, é necessário utilizar de um Check-list para o melhor aproveitamento de todos os recursos. Seguem então algumas dicas preciosas para este processo:

• Verificar se não existem Triggers vinculadas às tabelas que serão manipuladas. Triggers são muito úteis no processo casual, porém, quando há a necessidade de uma grande inserção de dados de uma só vez, não trabalhar com este recurso torna o processo mais ágil.

• O trabalho com sequenciais (Generators/Sequences) é algo indispensável no método convencional, no entanto, o consumo deste para cada registro também é algo caro para o processo de importação de muitos registros. Em casos de muita informação vale a pena obter o último registro de uma sequência e incrementá-lo por meio do Delphi, e só então após a importação, atualizar o banco com o valor do sequencial.

• Se houver a necessidade de realizar consultas antes de inserir algum registro no banco de dados, procure carregar os dados da tabela que será consultada diversas vezes em memória (via TClientDataSet). Isto irá poupar idas ao banco, melhorando o desempenho da execução. Não descarte a possibilidade de utilizar comandos do próprio ClientDataSet como Filter e Locate.

• Crie índices em memória para que o processo de manipulação e pesquisa sejam mais rápidos, isto pode ser feito simplesmente informando na propriedade IndexName do ClientDataSet o nome do campo que se deseja indexar.

• Comandos SQL já compilados fazem a diferença na hora de uma grande execução. Algo muito comum em Softwares legados Delphi, ou até mesmo em arquiteturas antigas é a utilização de comandos SQL do modo mais simples (diretamente injetado no código, como o exemplo da **Listagem 9**). Esta não é uma boa prática por diversos motivos como: dificuldade de manutenção sendo muito propícia a erros, difícil entendimento ao ler o código, além de que, toda vez que este comando necessitar ser enviado para o banco, ele será "Compilado" para só então ser executado e isto causa uma grande queda no desempenho das aplicações. Para sanar este tipo de problema, uma das formas é utilizar o comando SQL parametrizado em uma Query (**Listagem 10**). Deste modo, o processo de importação se torna mais elegante, de fácil manutenção, tendo assim um código mais limpo, além de ser executado de forma mais ágil.

Efetuando Log das operações

O log é um recurso vital na maioria dos aplicativos comerciais, com ele é possível manter as ações dos usuários no banco de dados salva para posterior conferência, porém, um grande impasse existente na hora de definir qual metodologia será utilizada para o salvamento do log é um dos maiores problemas deste tipo de necessidade.

Nota

Para não estender muito o artigo, parte do código-fonte não foi comentado, mas o exemplo se encontra disponível para download no portal DevMedia.

Listagem 9. Procedure Importar, exemplo de código e SQL dentro do código-fonte

```
procedure TDmExemploImportacao.Importar(
 ProCodigo,CliNome,CliDataNascimento,CliEndereco,CliBairro,
 CliCidade,CliUF,CliCEP,CliLimiteCompras,CliLimiteCredito : String);
var SQL : String;
begin
 SQL := 'INSERT INTO CLIENTES ('
    + ' PRO_CODIGO'
    + ',CLI_NOME'
    + ',CLI_DATA_NASCIMENTO'
    + ',CLI_ENDERECO'
    + ',CLI_BAIRRO'
    + ',CLI_CIDADE'
    + ',CLI_UF'
    + ',CLI_CEP'
    + ',CLI_LIMITE_COMPRAS'
    + ',CLI_LIMITE_CREDITO)'
    + ' VALUES ('
    + ProCodigo + ','
    + CliNome  + ','
    + CliDataNascimento + ','
    + CliEndereco    + ','
    + CliBairro + ','
    + CliCidade + ','
    + CliUF + ','
    + CliCEP + ','
    + CliLimiteCompras + ','
    + CliLimiteCredito + ')';

 sqlquery.SQL.Text := SQL;
 sqlquery.ExecSQL;
end;
```

Listagem 10. Exemplo de passagem de parâmetros

```
procedure TDmExemploImportacao.Importar;
begin
 sqlqryCliente.ParamByName('PRO_CODIGO').AsInteger := 0;
 sqlqryCliente.ParamByName('CLI_NOME').AsString  := '';
 sqlqryCliente.ParamByName('CLI_DATA_NASCIMENTO').AsDate := Now;
 sqlqryCliente.ParamByName('CLI_ENDERECO').AsString := '';
 sqlqryCliente.ParamByName('CLI_BAIRRO').AsString := '';
 sqlqryCliente.ParamByName('CLI_CIDADE').AsString := '';
 sqlqryCliente.ParamByName('CLI_UF').AsString := '';
 sqlqryCliente.ParamByName('CLI_CEP').AsString := '';
 sqlqryCliente.ParamByName('CLI_LIMITE_COMPRAS').AsFloat := 0;
 sqlqryCliente.ParamByName('CLI_LIMITE_CREDITO').AsFloat := 0;
 sqlqryCliente.ExecSQL;
end;
```

Partindo do pressuposto de que o registro de log não possa ser vinculado a usuário do Sistema Operacional, estação de trabalho, Triggers no banco, campos identificadores em cada tabela, possuímos então duas soluções para resolver este tipo de problema.

A primeira é utilizar o Trace do SQLConnection (**Figura 3**). Este uma vez habilitado, salva em arquivo todos os comandos SQL que são passados por ele durante uma execução, porém, o arquivo salvo por este Trace não aceita customizações em seu formato, o que causa um desconforto no momento de conferência.

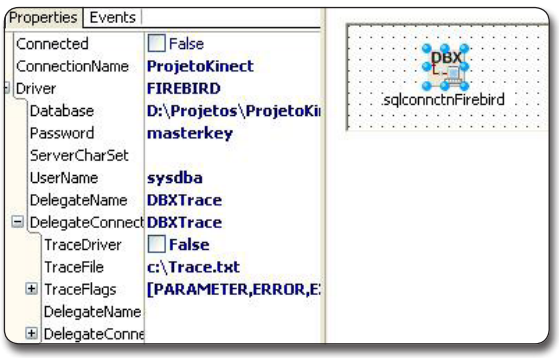


Figura 3. Trace via SQLConnection

Uma outra maneira de efetuar o processo de log é utilizar um conjunto de recursos disponíveis no dbExpress. Para isso, utiliza-se os eventos do componente DataSetProvider, que possui seu funcionamento semelhante ao de um Trigger de banco de dados, no entanto, suas ações ocorrem diretamente em seus eventos.

Mantendo o Log desta maneira o código fica desacoplado das regras de negócio, banco de dados, estação de trabalho e etc. Isto torna esta abordagem bastante genérica e ampla, onde com as devidas customizações, pode-se incluir qualquer tipo de informação.

Vale lembrar que o Log através do Trace também pode ser feito na arquitetura Cliente-Servidor utilizando o DataSnap.

Conclusão

Neste artigo abordamos diversos temas que são comuns no cotidiano de um desenvolvedor, tais como a manipulação de tabelas de um banco de dados, onde construímos uma estrutura sem a utilização direta de comandos SQL, mantendo a capacidade de trabalhar com múltiplos bancos de dados. Como consequência disto temos um dinamismo na hora de efetuar manutenção dos aplicativos, tornando cada vez mais fácil o trabalho com aplicativos multi-banco.

Também foi abordada a questão de otimizações e boas práticas na utilização de comandos SQL dentro do código, visando a melhor forma de se trabalhar com esta perspectiva. Geralmente este é um assunto bastante importante, pois um código bem escrito, além de facilmente compreendido, também proporciona uma rápida manutenção e consequentemente, um rendimento e um melhor aproveitamento de tempo. Ainda na questão de otimizações de uso de comandos SQL, vale a pena ressaltar que a abordagem aqui

contida é apenas um aspecto do dbExpress. Os artigos anteriores mostraram uma característica bastante abrangente de como se utilizar os componentes para seu melhor uso e estruturação de acesso a dados.

Por fim, falamos aqui um pouco sobre o trabalho com o processo de Log, processo este que é comumente utilizado de diversas formas. Com isso, finalizamos esta série de artigos que aborda recursos do dbExpress visualizando boa parte de seus recursos, o que facilmente pode proporcionar a criação de aplicações robustas, escaláveis e o mais importante, com qualidade. Até a próxima.

Alexandre Borges
contato@all-s.info
Desenvolvedor, instrutor e consultor: Delphi, C#, iOS, Maker. Graduado em gestão de TI, certificado Delphi Developer, Master, Microsoft MCP. Instrutor Oficial da Embarcadero em Goiânia.

Generate Data
http://www.generatedata.com/#generator

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

CURSOS ONLINE

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

.net magazine

MAIS de 25 CURSOS DISPONÍVEIS

CONHEÇA OS CURSOS MAIS RECENTES:

- Curso .net Entity Framework - Loja Virtual
- Novidades do Visual Studio 2010
- Componentes DevExpress
- Crie um chat utilizando Asp.Net 3.5 e Linq
- Novidades do C# (csharp) 3.0
- POO com C#
- Microsoft Visual Studio Tools for Office (VSTO)

Para mais informações :
www.devmedia.com.br/curso/netmagazine
(21) 3382-5038

DevMedia

Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8.000 artigos, além de palestras e vídeos com formações completas.

TODAS as vídeo aulas que você sempre quis ver a partir de

R$ 1,20

ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!

Saiba mais sobre o Sistema de Créditos!

www.devmedia.com.br/creditos

DevMedia

TENHA ACESSO ILIMITADO A TODO CONTEÚDO DO SITE DEVMEDIA

SEJA UM ASSINANTE MVP!

Tenha acesso garantido a qualquer tema que você possa necessitar!

VIDEOS -AULAS

O assinante MVP acessa todas as vídeo-aulas do site DevMedia. Nenhum outro produto dá essa possibilidade. São mais de **7.000 video-aulas.**

CURSOS

Nosso site possui atualmente **+ de 250 cursos online** com a qualidade que você já conhece e confia. Com o MVP você terá acesso a todos eles.

REVISTAS

Assinante MVP acessa todas as nossas revistas na versão digital, incluindo todas as edições anteriores. Quando pintar aquela dúvida você terá mais de **3.500 artigos para consultar!**

ASSINATURA MVP MOST VALUABLE PROFESSIONAL

A assinatura MVP te dá a oportunidade de aumentar seu conhecimento em diversos assuntos, com qualidade e economia.

Invista na sua carreira se torne um profissional ainda mais completo!

POR APENAS R$59,90 POR MÊS

DevMedia

Acesse: www.devmedia.com.br/mvp